به نام خدا

دانشگاه تهران
دانشکده‌ی مهندسی برق و کامپیوتر

ژنیک

گزارش پروژه ۲

نام و نام خانوادگی	سید علی طباطبایی آل طه	ملیکه احقاقی
شماره‌ی دانشجویی	۸۱۰۱۹۴۴۶۲	۸۱۰۱۹۴۲۵۴
تاریخ ارسال گزارش	۹۷/۱/۲۸	

1-1 چکیده

در طول این پروژه با بهره گیری از الگوریتم ژنتیک به حل مساله ای در حوزه ی job scheduling پرداخته می شود. هدف از حل این مساله برنامه ریزی انتخاب واحد یک دانشکده است. به این صورت که آموزش باید پیش از شروع هر ترم دروس مختلفی را به اساتید مرتبط محول کرده و در زمان های مشخصی آن ها را اراءه کند. به این صورت که تداخلی در دروس اراءه شده توسط یک استاد و هم چنین دروس دانشجویان یک دوره رخ ندهد. این مساله را با مساله ی ژنتیک مدل سازی می کنیم و با طراحی دو تابع crossover و mutation به فرم مطلوب سعی داریم به بالاترین fitness که در این مساله رسیدن به بالاترین میزان خوش حالی در دانشجویان هست برسیم.

1-2 ارائه‌ی روش

درتوضیح نحوه ی پیاده سازی این الگوریتم مراحلی را طی می نماییم.

```
if __name__ == '__main__':
    start = time.time()
    read_inputs()
    population = generate_population()
    sort(population)
    generation = 1
    counter = 0
    max_fitness = population[0][2]
    while time.time() - start < 120 and counter < 1000:
        population = evolve(population)
        sort(population)
        fitness = population[0][2]
        if fitness - max_fitness < 50:
            counter += 1
        else:
            counter = 0
        if fitness > max_fitness:
            max_fitness = fitness
        generation += 1
    print('max_fitness after %isec and %i generation: %i' % (time.time() - start, generation, max_fitness))
    print_final_output(population[0])
```

Fig1-The main body of the algorithm

*در پیاده سازی بدنه ی اصلی تابع main مراحل پیاده سازی الگوریتم قابل مشاهده است.

- در تابع اولیه read_inputs() با دریافت اطلاعات از کاربر ورودی ها را ذخیره می نماییم. از دو لیست slots و courses برای نگه داری داده ها استفاده می کنیم. slot لیستی از زوج مرتب های (day, time) به عنوان اسلات هایی هست که درس ها در آن زمان ها می توانند اراءه شوند. لیست courses داده های id- happiness - professors و sadness را برای دروس مختلف نگهداری می نماید. با دریافت ورودی ها از کاربر لیست slot ها و سپس لیست courses ساخته می شود.
- در مرحله ی بعدی با اجرای تابع generate_population() جمعیت اولیه ساخته می شود. جمعیت اولیه ی انتخابی ما ۱۵۰ در نظر گرفته شده است. برای ساخت جمعیت اولیه لیستی از schedule ها ساخته می شود. هر schedule نقش یکی از کروموزوم های جمعیت اولیه را بازی می کند. اطلاعاتی که در هر schedule نگهداری می شود شامل لیستی از کلاس هاست که در واقع این لیست نمایانگر کروموزوم های ما خواهد بود. در کنار آن آرایه ای از درس های قابل اراءه در آن schedule نگه داری می شود. هم چنین یک fitness در هر schedule قرار دارد. توجه شود که هنگام دریافت ورودی از کاربر یک سری درس وجود دارد که استادی آن را اراءه نمی دهد و در ابتدا آن ها را از لیست دروس حذف می نماییم.(هر درس همانطور که گفته شد شامل id -

professors -happiness و sadness می شود.) در این مرحله به تعداد population size schedule, ایجاد می نماییم. سپس آن ها را به جمعیت جدید اضافه می نماییم.برای ساخت هر schedule ابتدا به صورت اتفاقی یکی از دروس را انتخاب می کنیم. هم چنین یک اسلات نیز به صورت رندوم در نظر می گیریم. در هر اسلات توجه می کنیم که بین استاد هایی که می توانند این درس را اراءه دهند استادی در همان اسلات درس دیگری اراءه ندهد. در این صورت آن استاد را از لیست professor های آن درس حذف می کنیم. اگر استادی برای آن کلاس وجود داشت یک استاد به صورت رندوم برای آن کلاس انتخاب می نماییم. اگر استادی وجود نداشت به تعداد REPEATS که آن را ۵ انتخاب نموده ایم برای همین course به صورت اتفاقی اسلات و استاد انتخاب می نماییم. هر بار چه این درس ساخته شود چه نشود بعد این عملیات آن درس را از لیست درس هایی که در این schedule قرار دارند حذف می نماییم. (توجه شود که هر بار ابتدا در هر schedule یک deepcopy از لیست دروس قرار می دهیم.) تا زمانی این عملیات را ادامه می دهیم که پیمایش لیست دروس تمام شود. نهایتا تابع fitness را برای schedule محاسبه می نماییم. fitness حاصل از مجموع happiness دروس اراءه شده با کسر sadness های حاصل از تداخل ها خواهد بود.

- در مرحله ی بعد در تابع sort(population)() جمعیت اولیه را بر حسب fitness ها مرتب سازی می نماییم. که در این صورت ایندکس صفر لیست حاصل دارای بالاترین fitness خواهد بود.
- با قرار دادن دو محدودیت روی مساله که اولا زمان فراتر از ۱۲۰ ثانیه نشود و دوم اینکه عدد counter بیش از ۷۰۰ نشود به اجرای حلقه ی مساله ادامه می دهیم.(counter در واقع تعداد دفعاتی است که اختلاف fitness فعلی با ماکسیمی از آن که تا این لحظه به دست آمده کم تر از ۳۰ شود.)
- هر بار population را به عنوان ورودی به تابع evolve() می دهیم. این تابع مسءول ایجاد crossover در جهت ایجاد جهش در جمعیت ورودی است. در ابتدا به تعداد elite_schedules که ما در این مساله ۱۵ در نظر گرفته ایم از اول population قبلی عینا کپی می کنیم. در میان schedule های باقی مانده شروع به تشکیل population نماییم. یک ثابت به نام cross-over rate در نظر می گیریم. هر بار عدد رندومی ایجاد می نماییم در صورتی که این عدد از این رنج کوچکتر باشد cross-over انجام خواهد شد. در غیر این صورت همان schedule iام را عینا وارد population جدید می نماییم. در این مرحله برای پیدا کردن دو schedule که قرار است cross-over شود از تابع select_tournament_population() استفاده می نماییم. در این تابع به تعداد TOURNAMENT_SIZE که آن را ۳۷ در نظر گرفته ایم به صورت رندوم از schedule های جمعیت قبلی را انتخاب و سپس مرتب سازی می نماییم. در بین لیست حاصل بالاترین population ای با بالاترین fitness را به عنوان schedule1 انتخاب می نماییم. به همین صورت schedule2 را انتخاب می کنیم و بین این دو cross-over می زنیم. در cross-over یک schedule ایجاد می نماییم. برای تشکیل آن به تعداد طول ماکسیمم دو schedule حلقه خواهیم داشت که با احتمال نیم هر بار یک درس از یکی از آن دو انتخاب می نماییم. پس schedule نهایی ایجاد خواهد شد. کلاسی که اضافه می شود نباید با کلاس های افزوده شده تداخلی داشته باشد.
- حال از تابعی به نام add_new_class() استفاده می نماییم که نقش mutation را ایفا می کند. در این حالت از بین course هایی که کلاسی برای آن ها اراءه نشده است کلاس هایی را که می توانیم اضافه می نماییم و به همین ترتیب ادامه می دهیم.

1-3 ارائه‌ی نتایج

- **بررسی population size**

در این بخش برای بررسی اندازه ی نسبتا بهینه برای جمعیت اولیه با ثابت قرار دادن نسبی دیگر پارامتر ها به population مقادیر زیر را می دهیم. خروجی ها در ادامه آورده شده است.

```
Malikeh-2:ai-spring-97-2 malikeh$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 12278
max_fitness after 120sec and 292 generation: 12770

real    2m0.263s
user    1m59.222s
sys     0m0.512s
Malikeh-2:ai-spring-97-2 malikeh$
```

Fig2-Population size = 300

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 12064
max_fitness after 120sec and 327 generation: 12762

real    2m0.076s
user    1m59.413s
sys     0m0.349s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig3-Population size = 150

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 100
repeats = 5
crossover rate = 0
tournament size = 25
number of elite schedules = 10
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 12176
max_fitness after 120sec and 502 generation: 12532

real    2m0.069s
user    1m59.482s
sys     0m0.308s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig3-Population size = 100

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 50
repeats = 5
crossover rate = 0
tournament size = 12
number of elite schedules = 5
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 12193
max_fitness after 82sec and 706 generation: 12434

real    1m22.218s
user    1m21.747s
sys     0m0.220s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig4-Population size = 50

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 30
repeats = 5
crossover rate = 0
tournament size = 7
number of elite schedules = 3
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 11951
max_fitness after 49sec and 703 generation: 12222

real    0m49.618s
user    0m49.394s
sys     0m0.130s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig5-Population size = 30

- **بررسی elite number**

در این بخش برای بررسی اندازه ی نسبتا بهینه برای elite number با ثابت قرار دادن نسبی دیگر پارامتر ها به elite number مقادیر زیر را می دهیم. خروجی ها در ادامه آورده شده است.

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input2.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 9408
max_fitness after 120sec and 429 generation: 9576

real    2m0.350s
user    1m59.827s
sys     0m0.281s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input3.txt
```

Fig6-elite number = 15

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 7
generations with little diffrence = 700
fitness diffrence = 30
------------------------------------------------------------
fitness of fittest schedule of initial population: 12256
max_fitness after 120sec and 329 generation: 12858

real    2m0.180s
user    1m59.998s
sys     0m0.093s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig7-elite number = 7

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 30
generations with little diffrence = 700
fitness diffrence = 30
------------------------------------------------------------
fitness of fittest schedule of initial population: 12181
max_fitness after 120sec and 390 generation: 12714

real    2m0.263s
user    1m59.951s
sys     0m0.158s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig8-elite number = 30

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 75
generations with little diffrence = 700
fitness diffrence = 30
------------------------------------------------------------
fitness of fittest schedule of initial population: 12305
max_fitness after 120sec and 633 generation: 12542

real    2m0.073s
user    1m59.933s
sys     0m0.078s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig9-elite number = 75

- **بررسی ۴ ورودی نمونه به پیاده سازی نهایی**

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input1.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 12064
max_fitness after 120sec and 327 generation: 12762

real    2m0.076s
user    1m59.413s
sys     0m0.349s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$
```

Fig10-input1

```
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input2.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 9408
max_fitness after 120sec and 429 generation: 9576

real    2m0.350s
user    1m59.827s
sys     0m0.281s
AliTabatabaeis-MacBook-Pro:ai-spring-97-2 alitabatabaeiat$ time python3 Main.py < inputs/input3.txt
```

Fig11-input2

```
Malikeh-2:ai-spring-97-2 malikeh$ time python3 Main.py < inputs/input3.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 15704
max_fitness after 120sec and 171 generation: 17192

real    2m0.211s
user    1m58.243s
sys     0m0.718s
```

Fig12-input3

```
Malikeh-2:ai-spring-97-2 malikeh$ time python3 Main.py < inputs/input4.txt
population size = 150
repeats = 5
crossover rate = 0
tournament size = 37
number of elite schedules = 15
generations with little diffrence = 700
fitness diffrence = 30
----------------------------------------------------------
fitness of fittest schedule of initial population: 5983
max_fitness after 120sec and 305 generation: 6047

real    2m0.371s
user    1m57.376s
sys     0m0.944s
```

Fig13-input4

1-4 تحلیل نتایج

در حل این مساله با کمک الگوریتم ژنتیک فاکتور های زیادی برای بهینه سازی جواب وجود دارد .اگر مرحله به مرحله پیش برویم باید گفت که اولین فاکتور قابل بررسی اندازه ی جمعیت اولیه است که با توجه به ورودی های نمونه و ثابت قرار دادن نسبی پارامتر های دیگر با عدد۱۵۰ به خروجی نسبتا مطلوبی از لحاظ زمانی و fitness رسیدیم. توجه شود که انتخاب اولیه ی جمعیت باید به گونه ای باشد که اولا دقت کافی حاصل از بالا بودن نمونه های کروموزوم را داشته باشید و دوما با ایجاد تعداد محدود تری از generation در زمان معقولی به همگرایی نسبی برسد. نکته ی بعدی که در طراحی ما وجود داشت انتخاب محدودیتی در تکرار عمل ()evolve بود. توجه شود که محدودیت زمانی که از جانب صورت سوال اعمال شده بود لحاظ شده است اما غیر از آن شرط همگرایی را با عدد ۷۰۰ نشان داده ایم. انتخاب این عدد کاملا experienced است. بنابراین که به طور حدودی generation حاصل چه میزان خواهد بود و چه عدد نسبتا خوبی ما را در این محدوده ی زمانی همگرایی خواهد رساند. نکته ی قابل توجه بعدی در این مساله طراحی تابع cross-over بود. اول این که select_tournament باید تعداد مناسبی برای schedule های رندوم در نظر بگیرد که ما به صورت تجربی این مقدار را ۲۵٪ از جمعیت اولیه در نظر می گیریم. یک مساله ی اساسی برای انتخاب مناسب elite number این است که با بزرگ بودن آن جهشی نخواهیم داشت و خروجی حول یک ناحیه مرکزیت پیدا می کند. پس با توجه به خروجی های به دست آمده و تعداد جمعیت اولیه این عدد را برابر با ۱۰٪ از جمعیت اولیه در نظر گرفتیم. در پیاده سازی این الگوریتم روش دیگری را برای cross-over در نظر گرفتیم در نهایت این روش را پیاده سازی نمودیم.(هر دو روش در git repository موجود است) در روش دوم که سرعت و fitness نیز بالاتر است لیست دروس دو درس را concat می کنیم. هر بار به صورت رندوم یکی از این دروس را انتخاب و در صورت عدم تداخل قبلی ان را به لیست اضافه می نماییم. توجه شود که در انتخاب بین یکی از دو schedule1 و schedule2 اگر معیار احتمالی را fitnessi/fitness1+fitness2 در نظر بگیریم با محاسبه ی تجربی عددی حدود ۰.۴۹ حاصل شد پس اولویت یکسانی بین انتخاب بین دو schedule در نظر گرفتیم.از طرفی با انتخاب اختلاف fitness به اندازه ی ۳۰ به صورت تجربی سعی کردیم تا زمانی که این اتفاق بیش از ۷۰۰ بار اتفاق بیافتد ()evolve اجرا شود و در این صورت همگرایی برقرار خواهد شد.

1-5 جمع‌بندی و نتیجه‌گیری

مساله ی نخست که در ساخت جمعیت اولیه وجود دارد انتخاب optimal population size برای آن است. افزایش population size دقت الگوریتم ژنتیک را بالا می برد. نتایج تحقیقات نشان می دهد که هر آن چه که population size افزایش یابد احتمال رسیدن به کروموزوم بهینه بالاتر خواهد بود. سایز بهینه در انتخاب جمعیت اولیه بایستی که بین تعداد کمی از generation های همگرا و دقت حاصل از افزایش جمعیت تعادل برقرار کند.

از طرفی در مدل سازی حل مساله در فضای الگوریتم ژنتیک بایستی توابع cross-over و mutation به گونه ای طراحی شود که به صورت تجربی بر حسب فضای نمونه ای که در آن کار می کنیم باعث همگرایی جمعیت به سمتی شود که به بالاترین fitness برسد. پارامترها به صورت learning محاسبه می شوند و بایستی جهش های ابتدایی و نهایتا همگرایی جمعیت را در پی داشته باشند. این بررسی ها در زمینه ی توابع select_tournament() و elite_number هم چنین نحوه ی پیاده سازی cross-over و نهایتا mutation هایی که انجام می شود از دو طریق expert و learning انجام شود.