Genetic Scheduling

David Monteleone, Tomer Shemesh,
Kelly Shiptoski

Problem Statement

- Students at Drexel need a method for creating class schedules.
- Needs to be accurate
- Needs to be fast
- We wanted a program that built a near optimal schedule for us.
- Design and implement a program that can solve the problem for us!

Original Idea

- Use a neural network
- Wasn’t the easiest solution to get working
- HW4 gave us a new insight to solving the problem with genetic algorithms.
- Experimented with prompting the user on each generation for input to skew the results.
    - Unnecessary and too slow!

Problem Solution

- Use a genetic algorithm to determine a near optimal schedule.
- Allow for a *self learning* scheduling application
- Allow for user preferences to skew the scheduler
- This is described in detail later in the presentation.

Our Approach

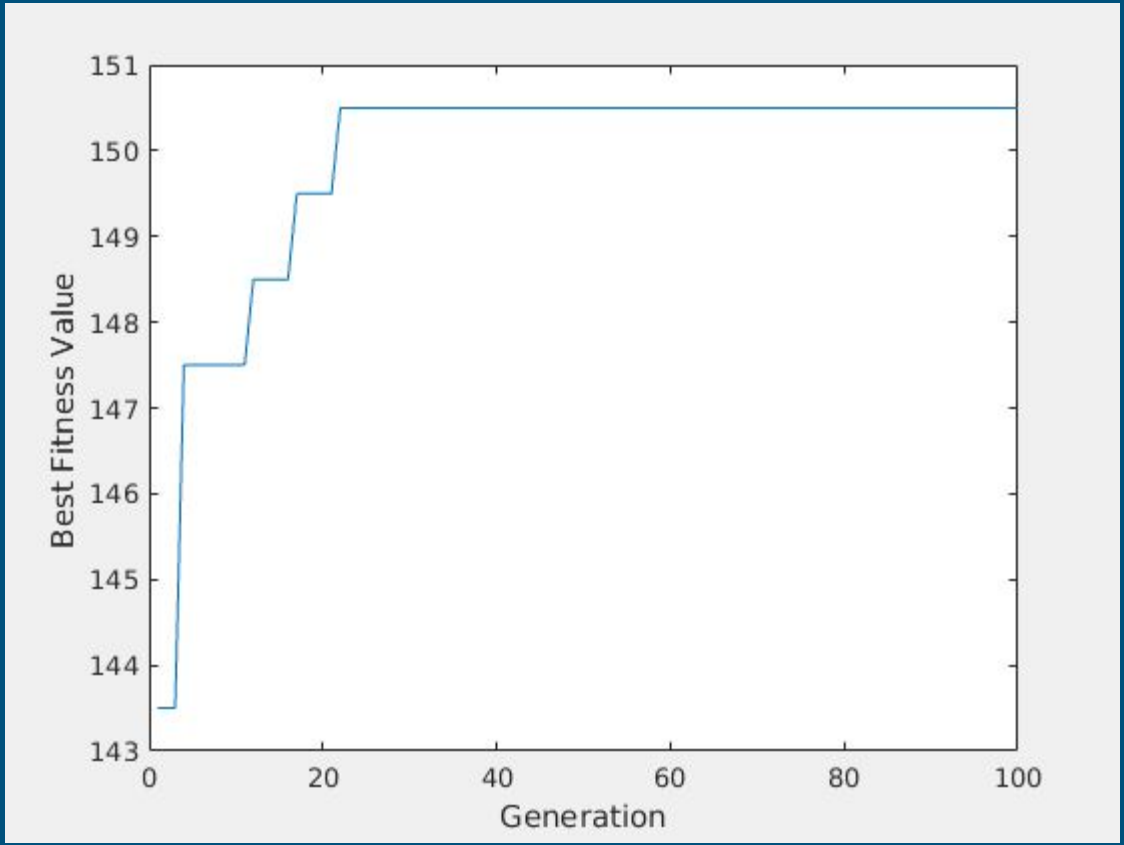
- Utilize MATLAB for its rapid prototyping
- Utilize a previously built Drexel WebTMS Ruby API (Just provides class info)
  - Made by Tomer
- Use a specialized genetic algorithm approach:
  - Have user defined preferences (online, tightly packed, morning, days we want off)
  - Generate random permutations of user specified classes
  - Crossover the top 6 schedules to create 3 more schedules to add to the population
  - Get the fitness scores of the children and place them into the population (sorted)
  - Perform for 100 generations (For speed!)
  - The top scored fitness function at the end is the best schedule

Our Results

- Excellent self learning scheduler!
- Distinct schedules based on user defined preferences!
- Let’s see some examples!

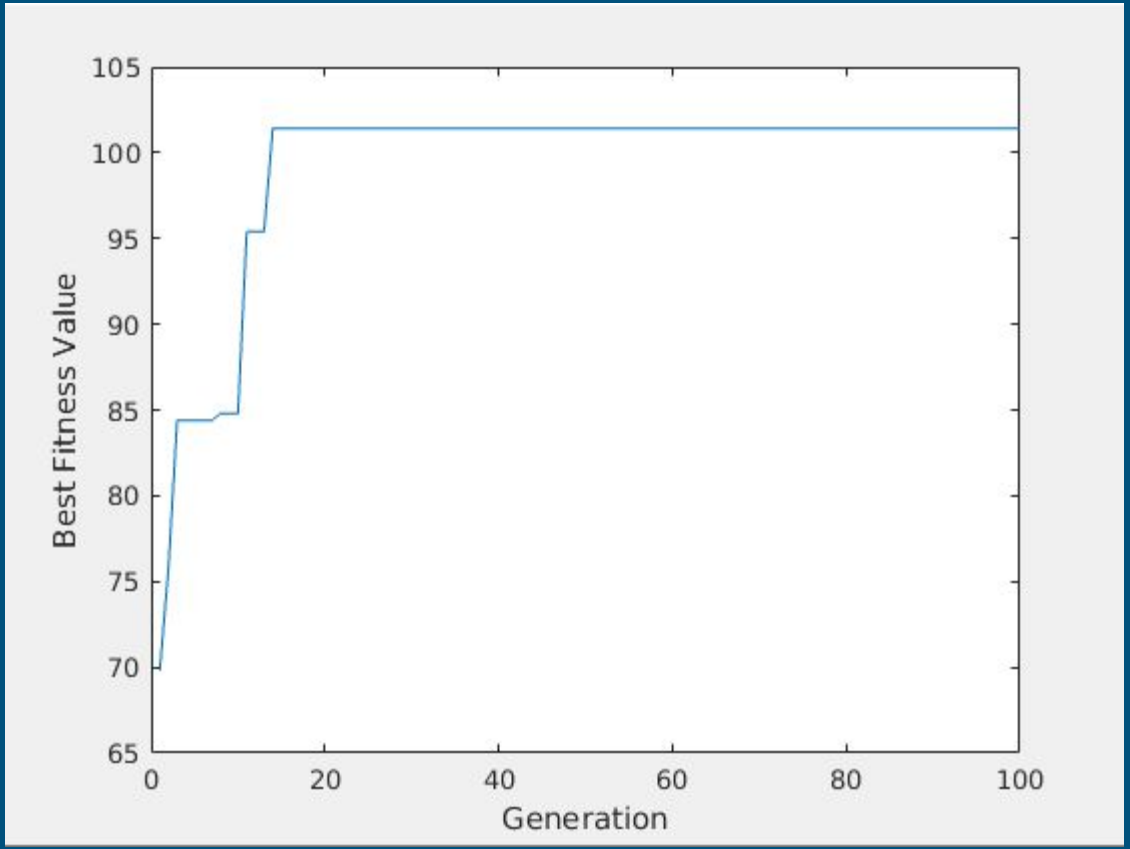
```
preferences = struct();
preferences.online_classes = 0;
preferences.tightly_packed = 10;
preferences.morning_classes = 10
preferences.no_classes = ['F'];
```

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Online
8:00am	ECON201	CHEM102	ECON201				
8:30am	ECON201	CHEM102	ECON201				
9:00am	ECON201	CHEM102	ECON201	ENGR102			
9:30am	ECON201	CHEM102	ECON201	ENGR102			
10:00am				ENGR102			
10:30am				ENGR102			
11:00am	ENGR102	CHEM102					
11:30am	ENGR102	CHEM102					
12:00pm	CHEM102		CHEM102		CHEM102		
12:30pm	CHEM102	CS283	CHEM102	CS283	CHEM102		
1:00pm		CS283		CS283			
1:30pm		CS283		CS283			
2:00pm							
2:30pm							
3:00pm							
3:30pm							
4:00pm							
4:30pm							
5:00pm							
5:30pm							
6:00pm							
6:30pm							
7:00pm							
7:30pm							
8:00pm							
8:30pm							
9:00pm							
9:30pm							
10:00pm							


Best Fitness Value
151
150
149
148
147
146
145
144
143
0
20
40
60
80
100
Generation

```
preferences = struct();
preferences.online_classes = 10;
preferences.tightly_packed = 10;
preferences.morning_classes = 0;
preferences.no_classes = ['W'];
```

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Online
8:00am							ECON201
8:30am							
9:00am							
9:30am							
10:00am							
10:30am							
11:00am							
11:30am							
12:00pm			CHEM102				
12:30pm			CHEM102				
1:00pm		ENGR102	CHEM102				
1:30pm		ENGR102	CHEM102				
2:00pm	ENGR102	ENGR102	CHEM102				
2:30pm	ENGR102	ENGR102	CHEM102				
3:00pm							
3:30pm							
4:00pm	CHEM102		CHEM102		CHEM102		
4:30pm	CHEM102		CHEM102		CHEM102		
5:00pm							
5:30pm							
6:00pm							
6:30pm				CS283			
7:00pm				CS283			
7:30pm				CS283			
8:00pm				CS283			
8:30pm				CS283			
9:00pm				CS283			
9:30pm							
10:00pm							


105
100
95
90
85
80
75
70
65
Best Fitness Value
0
20
40
60
80
100
Generation

Problem Representation

- MATLAB structs and cell arrays
  - Represent the population and list of current schedules

```
schedules =

    {7x7 cell}    [61.8000]
    {7x7 cell}    [59.5000]
    {7x7 cell}    [57.8000]
    {7x7 cell}    [54.8000]
    {7x7 cell}    [43.3000]
    {7x7 cell}    [41.5000]
    {7x7 cell}    [39.3000]
    {7x7 cell}    [38.3000]
    {7x7 cell}    [     34]
```

```
K>> schedules{1}

ans =

    [1x1 struct]
    [1x1 struct]
    [1x1 struct]
    [1x1 struct]
    [1x1 struct]
    [1x1 struct]
    [1x1 struct]
```

REST API Struct

```
K>> schedules{1}{1}

ans =

                    id: 3693
                   crn: '21354'
              class_id: 'ECON201'
      instruction_type: 'Lecture'
    instruction_method: 'Face To Face'
               section: '005'
            begin_time: '2000-01-01T12:00:00.000-05:00'
              end_time: '2000-01-01T13:50:00.000-05:00'
            max_enroll: 60
         current_enroll: 60
               credits: 4
                campus: 'University City'
      section_comments: 'Waitlist capabilities until Tuesday prior to class beginning…'
         textbook_link: 'http://drexel.bncollege.com/webapp/wcs/stores/servlet/TBListView?cm_mmc=RI-_-457-_-1-_…'
              building: 'GHALL'
                  room: '033'
                  term: 'Winter'
             term_type: 'Quarter'
             term_year: '15-16'
            created_at: '2015-12-19T05:02:12.601-05:00'
            updated_at: '2016-01-26T13:18:51.047-05:00'
      days_time_string: 'TR 12:00pm-1:50pm'
            professors: [1x1 struct]
```

Crossover

- Each child must be a permutation of a combination of the parents
- Defined as follows:
  - For each class position in the child:
    - Select one of two parents to copy that class position
    - Repeat for each class position
    - Give a random chance to mutate the child afterwards
- Built off of the cryptograph crossover discussed in lecture
- Other methods attempted:
  - Choose a random class in parent 1 and swap it with parent 2's into the child (Slow!)
  - Random permutations (Incredibly slow and inefficient)

Mutation

- 33% chance at a random mutation
  - Selects a random class and replaces with another random section of that class
- Higher rates resulted in:
  - Random schedules
  - Less stable ascent
- Lower rates
  - Quicker plateauing of the best schedule.

```
Parent1
ECON201 Lecture M 6:00pm-9:50pm
CS432 Lecture W 6:00pm-8:50pm
CS283 Lecture TR 12:30pm-1:50pm
CHEM102 Lecture MWF 12:00pm-12:50pm
CHEM102 Lab W 10:00am-11:50am
CHEM102 Recitation/Discussion M 1:00pm-1:50pm
Parent2
ECON201 Lecture TR 12:00pm-1:50pm
CS432 Lecture W 6:00pm-8:50pm
CS283 Lecture R 6:30pm-9:20pm
CHEM102 Lecture MWF 4:00pm-4:50pm
CHEM102 Lab W 12:00pm-1:50pm
CHEM102 Recitation/Discussion W 10:00am-10:50am
Child
ECON201 Lecture TR 12:00pm-1:50pm
CS432 Lecture W 6:00pm-8:50pm
CS283 Lecture R 6:30pm-9:20pm
CHEM102 Lecture MWF 12:00pm-12:50pm
CHEM102 Lab W 10:00am-11:50am
CHEM102 Recitation/Discussion T 2:00pm-2:50pm
```

```
if randi(3,1,1) == 1
   classSwap = randi(size(child),1,1);
   nums = size(all_classes{classSwap},1);
   order_to_try = randperm(nums);
   for i = 1:nums
        if Fits(child,all_classes{classSwap}(order_to_try(i)))
            child{classSwap} = all_classes{classSwap}(order_to_try(i));
            break;
        end
   end
end
```

Fitness Function

- User preference based!
- Skews best schedules to morning, tightly compact, online, etc…
- Schedules are awarded “points” for satisfying the defined preferences
- Points are summed into a collective total score

$$F = F1 + F2 + F3 + \ldots + Fn$$

```
fitness = 0;

fitness = fitness + (preferences.online_classes * online_classes);
fitness = fitness + ((10-preferences.online_classes) * (size(classes,1) - online_classes))/2;

for i = 1:size(mapkeys,2)
    times = mapObj(char(mapkeys(i)));
    if times(3) ~= 0
        fitness = fitness + ((14-((times(2) - times(1))/100)) * preferences.tightly_packed)/10;
        fitness = fitness + ((14-((times(1)-800)/100)) * preferences.morning_classes)/10;
    end
end

for i = 1:size(preferences.no_classes)
    time = mapObj(char(preferences.no_classes(i)));
    if time(3) == 0
        fitness = fitness + 10;
    end
end
```

Live Demo

Comparison with Other Methods

- Brute Force and Depth First Search
- 7 classes, all with 4-49 sections available
- 24,300,864 possible combinations of schedules
- This would take weeks to find the best schedule:

Running at 1/10th a second per schedule

24,300,864 / 10 per second / 60 seconds / 60 minutes / 24 hours = 27.7 Days!

- Ours finds a great suboptimal schedule within a minute

Uses in Other Applications

- Scale up!
- Airport scheduling
- Train scheduling
- Genetic algorithms have proved to be an effective way to get a favorable suboptimal answer to relatively small datasets.

Conclusion

- This is an excellent method for creating student schedules.
- Small scale experiments prove it to be effective.
- Given more time we could build this into a mobile app. (lightweight!)
- We’ve shown a method for letting a program solve the problem instead of the programmers solving the problem directly!

Questions