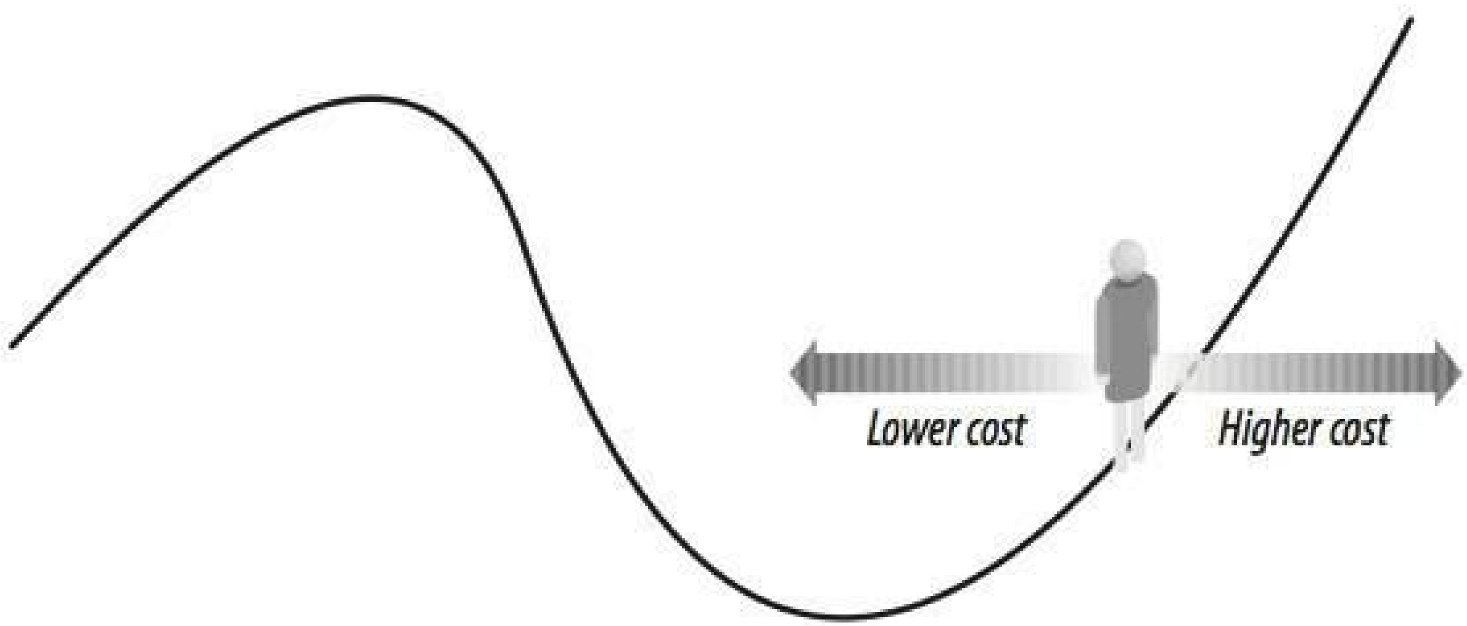
ENGR 212: Programming Practice

Week 14

Educated Optimization

- Random Searching
- **Hill Climbing**
- Genetic Algorithm

Hill Climbing



Hill Climbing

- Start with a random schedule and find all the neighboring schedules.
- In this case, that means finding all the schedules that have one person on a slightly earlier or slightly later flight.
- The cost is calculated for each of the neighboring schedules, and the one with the lowest cost becomes the new solution.
- This process is repeated <u>until none of the neighboring schedules improves the cost.</u>

Hill Climbing (Part 1)

```python
def hillclimb(domain, costf):
    # Create a random solution
    sol = [random.randint(domain[i][0], domain[i][1])
            for i in range(len(domain))]

    # Main loop
    while True:
        # Create list of neighboring solutions
        neighbors = []
        for j in range(len(domain)):
            # One away in each direction
            if sol[j] > domain[j][0]:
                neighbors.append(sol[0:j] + [sol[j] - 1] + sol[j + 1:])
            if sol[j] < domain[j][1]:
                neighbors.append(sol[0:j] + [sol[j] + 1] + sol[j + 1:])
```

Hill Climbing (Part 2)

```
    ...

    # See what the best solution amongst the neighbors is
    current = costf(sol)
    best = current
    for j in range(len(neighbors)):
        cost = costf(neighbors[j])
        if cost < best:
            best = cost
            sol = neighbors[j]

    if best == current:
        break

return sol
```

Hill Climbing

```
>>> s = optimization.hillclimb(domain,
optimization.schedulecost)

>>> optimization.schedulecost(s)

>>> optimization.printschedule(s)
```

Hill Climbing

Educated Optimization

- Random Searching
- Hill Climbing
- **Genetic Algorithm**

Genetic Algorithms

- Be elitist!
- Pick elites!
- Mutate elites!
- Crossbreed elites!

Rank solutions based on cost!

Solution	Cost
[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3]	4394
[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8]	4661
...	...
[0, 4, 0, 3, 8, 8, 4, 4, 8, 5, 6, 1]	7845
[5, 8, 0, 2, 8, 8, 8, 2, 1, 6, 6, 8]	8088

Mutation

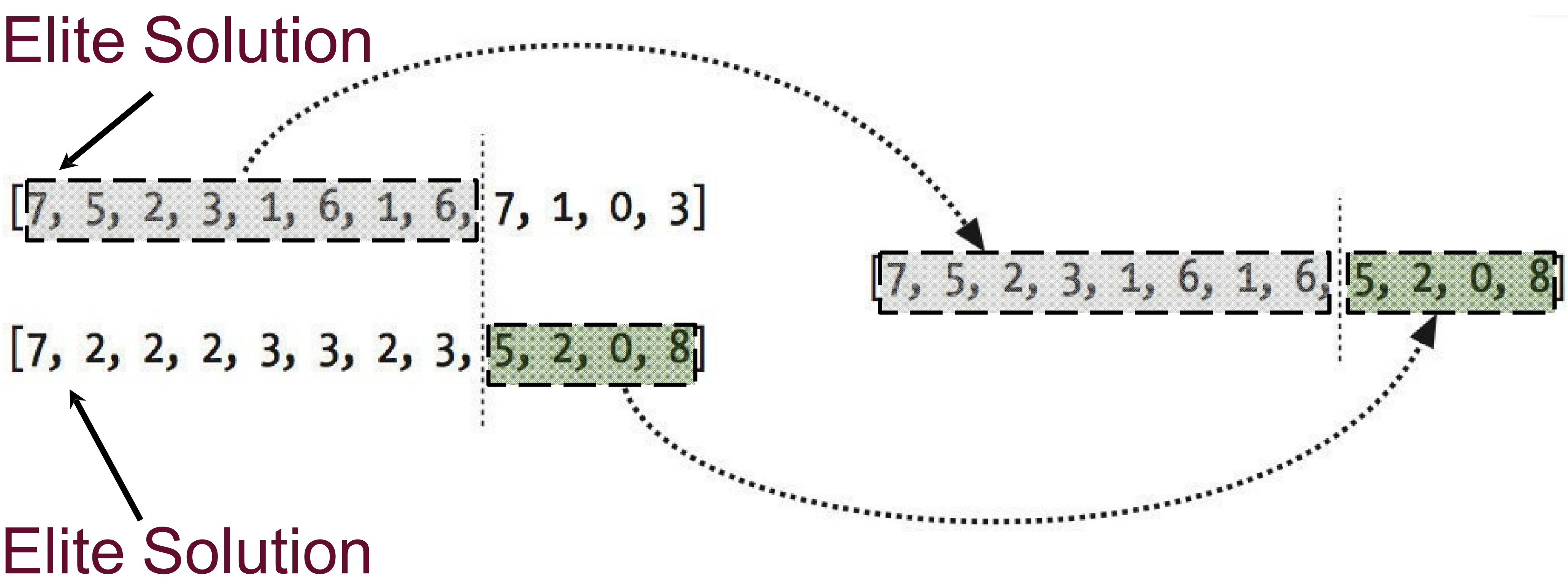
Elite Solution

[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3] → [7, 5, 2, 3, 1, 6, 1, 5, 7, 1, 0, 3]

[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8] → [7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 1, 8]

Elite Solution

Crossover


Elite Solution
[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3]
[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8]
Elite Solution
[7, 5, 2, 3, 1, 6, 1, 6, 5, 2, 0, 8]

Genetic Algorithms (Part 1)

```python
def geneticoptimize(domain, costf, popsize=50, step=1, mutprob=0.2,
                    elite=0.2, maxiter=100):
    # Mutation Operation
    def mutate(vec):
        i = random.randint(0, len(domain)-1)
        if random.random() < 0.5 and vec[i] > domain[i][0]:
            return vec[0:i] + [vec[i] - step] + vec[i + 1:]
        elif vec[i] < domain[i][1]:
            return vec[0:i] + [vec[i] + step] + vec[i + 1:]
        else:
            return vec[:]

    # Crossover Operation
    def crossover(r1, r2):
        i = random.randint(1, len(domain)-1)
        return r1[0:i] + r2[i:]
```

Genetic Algorithms (Part 2)

```
def geneticoptimize(domain, costf, popsize=50, step=1, mutprob=0.2,
                    elite=0.2, maxiter=100):
    ...
    ...

    # Build the initial population
    pop = []
    for i in range(popsize):
        vec = [random.randint(domain[i][0], domain[i][1])
               for i in range(len(domain))]
        pop.append(vec)

    # How many winners from each generation?
    topelite = int(elite * popsize)
```

Genetic Algorithms (Part 3)

```
def geneticoptimize(domain, costf, popsize=50, step=1, mutprob=0.2,
                    elite=0.2, maxiter=100):
    ...

    # Main loop
    for i in range(maxiter):
        scores = [(costf(v), v) for v in pop]
        scores.sort()
        ranked = [v for (s, v) in scores]

        # Start with the pure winners
        pop = ranked[0:topelite]

        # Add mutated and bred forms of the winners
        while len(pop) < popsize:
            if random.random() < mutprob:
                # Mutation
                c = random.randint(0, topelite-1)
                pop.append(mutate(ranked[c]))
            else:
                # Crossover
                c1 = random.randint(0, topelite-1)
                c2 = random.randint(0, topelite-1)
                pop.append(crossover(ranked[c1], ranked[c2]))

        # Print current best score
        print scores[0][0]

    return scores[0][1]
```

Genetic Algorithms - Practice

```
>>> s = optimization.geneticoptimize(domain, optimization.schedulecost)

>>> optimization.schedulecost(s)

>>> optimization.printschedule(s)
```

Optimizing for Preferences

Student dorm optimization

- 5 Dorms ['Zeus','Athena','Hercules','Bacchus','Pluto']
- 2 vacant spots in each.
- 10 students waiting for accommodation.
- How many possible combinations?

How to represent solution space?

- Think of our flight search solution representation.
- We can create a list of numbers, one for each student, where each number represents the dorm in which you’ve put the student.
- Does this representation enforce our constraint of having “only two students in each dorm” ?
- For example, a list of all zeros would indicate that everyone had been placed in Zeus, which isn’t a real solution at all.

How to represent solution space?

- So is it really bad?

  —*What happens when there are invalid solutions in the solution terrain?*

- Can we make the cost function return a very high value for invalid solutions?

  —*It's better not to waste processor cycles searching among invalid solutions.*

How to represent solution space?

- Represent solutions so that every one is valid.
- A valid solution is not necessarily a good solution; it just means that there are exactly two students assigned to each dorm.
- Think of every dorm as having two slots, so that in the example there are ten slots in total.
- Each student is assigned to one of the open slots—the first person can be placed in any one of the ten, the second person can be placed in any of the nine remaining slots, and so on.

Student dorm optimization (Part 1)

```
import random
import math

# The dorms, each of which has two available spaces
dorms=['Zeus','Athena','Hercules','Bacchus','Pluto']

# People, along with their first and second choices
prefs=[('Toby', ('Bacchus', 'Hercules')),
       ('Steve', ('Zeus', 'Pluto')),
       ('Karen', ('Athena', 'Zeus')),
       ('Sarah', ('Zeus', 'Pluto')),
       ('Dave', ('Athena', 'Bacchus')),
       ('Jeff', ('Hercules', 'Pluto')),
       ('Fred', ('Pluto', 'Athena')),
       ('Suzie', ('Bacchus', 'Hercules')),
       ('Laura', ('Bacchus', 'Hercules')),
       ('James', ('Hercules', 'Athena'))]
```

Student dorm optimization (Part 2)

```
# [(0,9),(0,8),(0,7),(0,6),...,(0,0)]
domain=[(0,(len(dorms)*2)-i-1) for i in range(0,len(dorms)*2)]

def printsolution(vec):
  slots=[]
  # Create two slots for each dorm
  for i in range(len(dorms)): slots+=[i,i]

  # Loopover  each students assignment
  for i in range(len(vec)):
    x=int(vec[i])

    # Choose the slot from the remaining ones
    dorm=dorms[slots[x]]
    # Show the student and assigned dorm
    print prefs[i][0], dorm
    # Remove this slot
    del slots[x]
```

Student dorm optimization (Part 3)

```
def dormcost(vec):
  cost=0
  # Create two slots for each dorm
  slots=[0,0,1,1,2,2,3,3,4,4]

  # Loopover  each student
  for i in range(len(vec)):
    x=int(vec[i])
    dorm=dorms[slots[x]]
    pref=prefs[i][1]
    # First choice costs 0, second choice costs 1
    if pref[0]==dorm: cost+=0
    elif pref[1]==dorm: cost+=1
    # Not on the list costs 3
    else: cost+=3

    # Remove selected slot
    del slots[x]

  return cost
```

Student dorm optimization

```
>>> reload(dorm)
>>> s = optimization.randomoptimize(dorm.domain, dorm.dormcost)
>>> print dorm.dormcost(s)
>>> dorm.printsolution(s)
>>> s = optimization.geneticoptimize(dorm.domain, dorm.dormcost)
>>> dorm.printsolution(s)
```