INF3490/INF4490 Exercise Solutions - Week 2

Ole Herman S. Elgesem

October 3, 2016

$\mathbb{P}$ marks the programming exercises, we strongly recommend using the python programming language for these. Exercises may be added/changed after publishing.

1 Representations

Recall all the representations that have been presented. Which mutation and recombination operators are compatible with which representations?

Answer:

- Binary representation
  - Bit-flip mutation
  - N-point and uniform crossover
- Integer representation
  - Random reset and creep mutation
  - N-point and uniform crossover
- Cardinal/enumerated/symbolic representations
  - Random reset mutation
  - N-point and uniform crossover
- Real-valued/Continuous representation
  - Uniform and Gaussian mutation
  - N-point, discrete uniform and arithmetic crossover
- Permutation representation
  - Swap, insert, scramble and invert mutation
  - Partially mapped, order, cycle and edge crossover
- Tree representation
  - Mutation by random replacement
  - Subtree swap mutation

2 Bit flip mutation

Given the binary chromosome with length 4, calculate the probability that no bits, one bit and more than one bit will be flipped in a bit-flip mutation with $p_m = \frac{1}{4}$.

Answer:
Probability of no mutation:

$$P(0) = \frac{3}{4}\frac{3}{4}\frac{3}{4}\frac{3}{4} = \frac{3^4}{4^4} \approx 32\%$$

Binomial probability:

$$P = \binom{n}{k} p^k (1-p)^{n-k} \tag{1}$$

Binomial coefficient:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

Where n is number of events, k is number of occurences wanted and p is the probability of a single event.

In this case we want exactly 1 mutation in 4 events, and the probability is 0.25:

$$\begin{aligned}
n &= 4 \\
k &= 1 \\
p &= p_m = 0.25 \\
P(1) &= \binom{4}{1} 0.25^1 (1-0.25)^{4-1} \\
P(1) &= \frac{4!}{1!(4-1)!} 0.25^1 (1-0.25)^{4-1} \\
P(1) &= \frac{4*3*2}{3*2} 0.25^1 (0.75)^3 \\
P(1) &= 4 * 0.25^1 (0.75)^3 \\
P(1) &= 0.75^3 \approx 42\%
\end{aligned}$$

This makes intuitive sense and we could also arrive at this result without using the general formula. For example, the probability of mutation (yes,no,no,no) is:

$$\begin{aligned}
P(1) &= \frac{1}{4}\frac{3}{4}\frac{3}{4}\frac{3}{4} \\
P(1) &= 0.25 * 0.75^3
\end{aligned}$$

And there are 4 variants, 4 places where the mutation can happen, so:

$$\begin{aligned}
P(1) &= 4 * (0.25 * 0.75^3) \\
P(1) &= 0.75^3 \approx 42\%
\end{aligned}$$

Finally, the probability of more than one mutation:

$$\begin{aligned}
P(2+) &= 1 - P(0) - P(1) \\
P(2+) &\approx 1 - 32\% - 42\% \\
P(2+) &\approx 26\%
\end{aligned}$$

$\mathbb{P}$ 3 Crossover

Given the sequences (2,4,7,1,3,6,8,9,5) and (5,9,8,6,2,4,1,3,7). Implement these algorithms to create a new pair of solutions:

a. Partially mapped crossover (PMX).

b. Order crossover.

c. Cycle crossover.

Answer:

3.a Partially mapped crossover

3.a.1 Output

```
Parents:
[2, 4, 7, 1, 3, 6, 8, 9, 5]
[5, 9, 8, 6, 2, 4, 1, 3, 7]
Children:
[5, 9, 7, 1, 3, 6, 4, 2, 8]
[3, 1, 8, 6, 2, 4, 7, 9, 5]
```

3.a.2 Source code

```python
#!/usr/bin/env python3
import random

def pmx(a,b, start, stop):
    child = [None]*len(a)

    # Copy a slice from first parent:
    child[start:stop] = a[start:stop]

    # Map the same slice in parent b to child using indices from parent a:
    for ind,x in enumerate(b[start:stop]):
        ind += start
        if x not in child:
            while child[ind] != None:
                ind = b.index(a[ind])
            child[ind] = x

    # Copy over the rest from parent b
    for ind,x in enumerate(child):
        if x == None:
            child[ind] = b[ind]

    return child

def pmx_pair(a,b):
    half = len(a) // 2
    start = random.randint(0, len(a)-half)
    stop = start + half
    return pmx(a,b,start,stop) , pmx(b,a,start,stop)

if __name__ == "__main__":
    a = [2,4,7,1,3,6,8,9,5]
    b = [5,9,8,6,2,4,1,3,7]
    c,d = pmx_pair(a,b)
    print("Parents:")
    print(a)
    print(b)
    print("Children:")
    print(c)
    print(d)
```

3.b Order Crossover

3.b.1 Output

```
Parents:
[2, 4, 7, 1, 3, 6, 8, 9, 5]
[5, 9, 8, 6, 2, 4, 1, 3, 7]
Children:
[9, 2, 4, 1, 3, 6, 8, 7, 5]
[7, 3, 8, 6, 2, 4, 1, 9, 5]
```

3.b.2 Source code

```
#!/usr/bin/env python3
import random

def order_xover(a,b, start, stop):
    child = [None]*len(a)

    # Copy a slice from first parent:
    child[start:stop] = a[start:stop]

    # Fill using order from second parent:
    b_ind = stop
    c_ind = stop
    l = len(a)
    while None in child:
        if b[b_ind % l] not in child:
            child[c_ind % l] = b[b_ind % l]
            c_ind += 1
        b_ind += 1
    return child

def order_xover_pair(a,b):
    half = len(a) // 2
    start = random.randint(0, len(a)-half)
    stop = start + half
    return order_xover(a,b,start,stop) , order_xover(b,a,start,stop)

if __name__ == "__main__":
    a = [2,4,7,1,3,6,8,9,5]
    b = [5,9,8,6,2,4,1,3,7]
    c,d = order_xover_pair(a,b)
    print("Parents:")
    print(a)
    print(b)
    print("Children:")
    print(c)
    print(d)
```

3.c Cycle Crossover

3.c.1 Output

```
Parents:
[2, 4, 7, 1, 3, 6, 8, 9, 5]
[5, 9, 8, 6, 2, 4, 1, 3, 7]
```

```
Children:
[2, 4, 7, 1, 3, 6, 8, 9, 5]
[5, 9, 8, 6, 2, 4, 1, 3, 7]
```

(These 2 parents have just one cycle, so the child will be the same as parent 1).

3.c.2 Source code

```python
#!/usr/bin/env python3
import random

def cycle_xover(a,b):
    child = [None]*len(a)
    while None in child:
        ind = child.index(None)
        indices = []
        values = []
        while ind not in indices:
            val = a[ind]
            indices.append(ind)
            values.append(val)
            ind = a.index(b[ind])
        for ind,val in zip(indices, values):
            child[ind] = val
        a,b = b,a
    return child

def cycle_xover_pair(a,b):
    return cycle_xover(a,b) , cycle_xover(b,a)

if __name__ == "__main__":
    a = [2,4,7,1,3,6,8,9,5]
    b = [5,9,8,6,2,4,1,3,7]
    c,d = cycle_xover_pair(a,b)
    print("Parents:")
    print(a)
    print(b)
    print("Children:")
    print(c)
    print(d)
```

Contact and Github

Corrections of grammar, language, notation or suggestions for improving this material are appreciated. E-mail me at **olehelg@uio.no** or use **GitHub** to submit an issue or create a pull request. The **GitHub repository** contains all source code for assignments, exercises, solutions, examples etc. As many people have been involved with writing and updating the course material, they are not all listed as authors here. For a more complete list of authors and contributors see the **README**.