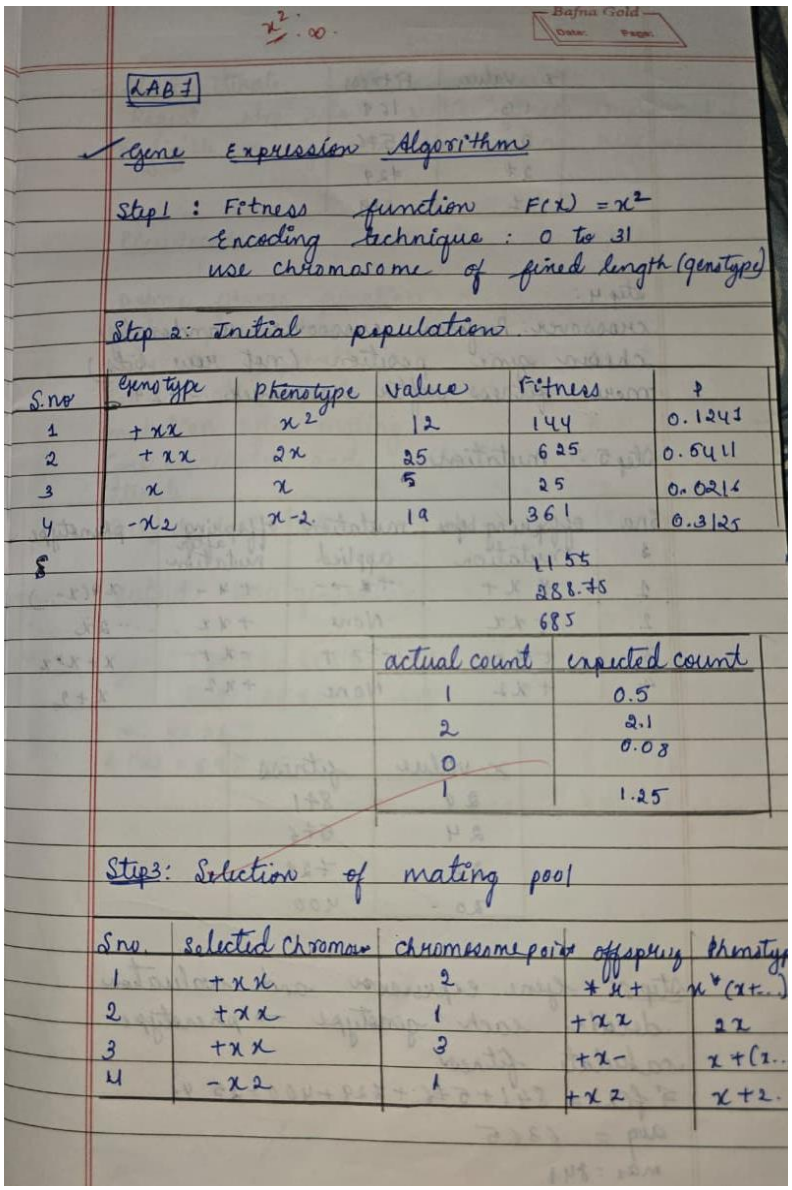
Program 7

Optimization via Gene Expression Algorithms The Travelling Salesman Problem (TSP) asks for the shortest possible route that visits a given set of cities exactly once and returns to the starting city. The provided text describes using a Genetic Algorithm to solve this by evolving city sequences (chromosomes) through selection, crossover, and mutation to minimize the total tour distance.

Algorithm:



$x^2 \cdot \infty$

LAB 7

Gene Expression Algorithm

Step 1 : Fitness function $F(x) = x^2$
Encoding technique : 0 to 31
use chromosome of fixed length (genotype)

Step 2: Initial population

S.no	Genotype	Phenotype	value	Fitness	P
1	+xx	x^2	12	144	0.1241
2	+xx	2x	25	625	0.5411
3	x	x	5	25	0.0216
4	-x2	x-2	19	361	0.3125
S				1155	
				288.75	
				685	

actual count	expected count
1	0.5
2	2.1
0	0.08
1	1.25

Step3: Selection of mating pool

S.no.	Selected Chromosome	chromosome pair	offspring	Phenotype
1	+xx	2	*x+	x*(x+..)
2	+xx	1	+xx	2x
3	+xx	3	+x-	x+(x..
4	-x2	1	+x2	x+2.

x value	Fitness
13	169
24	576
27	729
17	289

Step 4:

crossover: Perform crossover randomly chosen gene position (not raw bits) more fitness after crossover = 729

Step 5: mutation

Sno.	offspring before mutation	mutation applied	offspring after mutation	phenotype
1	* x +	+ ~~*~~ → -	+ * -	x*(x-...)
2	+ x x	None	+ x x	2x
3	+ x -	- → +	- x +	x + x*x
4	+ x 2	None	+ x 2	x + 2

x value	fitness
29	841
24	576
27	729
20	400

Step 6: Gene expression and evaluation
decode each genotype → phenotype
calculate fitness

$\Sigma f(x) = 841 + 576 + 729 + 400 = 254?$

avg = 6365

max = 841

Step 7: Iterate until convergence
Repeat step 3 to 6 until fitness Improvement is negligible or generator limit has reached

Pseudocode:

Define fitness function
Define parameters
generate population
Select mating pool
mutation after mating
Gene expression and evaluation
Iterate
Output Best value

output: 1000 generations

Genes: [29.53, 29.82, 29.84, 28.57, 15.09, 21.83, 23.03, 30.81, 28.51, 26.22]
x: 26.31
E(x) = 695.45

Code:

```
import random

def fitness(x):
    return x*x

def init_population():
    genes = ['+x', 'x', '2x', '-x']
    return random.sample(genes, 4)
```

```python
def express(gene, x):
    if gene == '+x': return x
    if gene == 'x': return x
    if gene == '2x': return 2*x
    if gene == '-x': return -x
    return x

def evaluate(pop):
    vals = [random.randint(1, 30) for _ in range(4)]
    phen = [express(pop[i], vals[i]) for i in range(4)]
    fit = [fitness(phen[i]) for i in range(4)]
    return vals, phen, fit

def select_mating(pop, fit):
    idx = sorted(range(4), key=lambda i: fit[i], reverse=True)
    return [pop[idx[0]], pop[idx[1]], pop[idx[2]], pop[idx[3]]]

def crossover(g1, g2):
    p = 1
    return g1[:p] + g2[p:], g2[:p] + g1[p:]

def mutate(gene):
    ops = ['+x', 'x', '2x', '-x']
    if random.random() < 0.3:
        return random.choice(ops)
    return gene

def gene_expression_algorithm(generations=10):
    pop = init_population()
    for gen in range(1, generations+1):
        vals, phen, fit = evaluate(pop)
        mating = select_mating(pop, fit)
        c1, c2 = crossover(mating[0], mating[1])
        c3, c4 = crossover(mating[2], mating[3])
        c1, c2, c3, c4 = mutate(c1), mutate(c2), mutate(c3), mutate(c4)
        pop = [c1, c2, c3, c4]
        best = max(fit)
        print(f"Generation {gen}: Fitness = {best}")
    return pop

result = gene_expression_algorithm()
print("\nFinal Genes:", result)
```

Output:

```
Generation 1: Fitness = 3600
Generation 2: Fitness = 400
Generation 3: Fitness = 484
Generation 4: Fitness = 324
Generation 5: Fitness = 841
Generation 6: Fitness = 900
Generation 7: Fitness = 784
Generation 8: Fitness = 441
Generation 9: Fitness = 625
Generation 10: Fitness = 576

Final Genes: ['x', 'x', '+x', '+x']
```