

ANT,GA,PSO,ES,G  
P,EP

# GA USING AND EXAMPLE

# THE KNAPSACK PROBLEM

## Definition

The KP problem is an example of a combinatorial optimization problem, which seeks for a best solution from among many other solutions. It is concerned with a knapsack that has positive integer volume (or capacity)  $V$ . There are  $n$  distinct items that may potentially be placed in the knapsack. Item  $i$  has a positive integer volume  $V_i$  and positive integer benefit  $B_i$ . In addition, there are  $Q_i$  copies of item  $i$  available, where quantity  $Q_i$  is a positive integer satisfying  $1 \leq Q_i \leq \infty$ .

Let  $X_i$  determines how many copies of item  $i$  are to be placed into the knapsack. The goal is to:

Maximize

$$\sum_{i=1}^N B_i X_i$$

Subject to the constraints

$$\sum_{i=1}^N V_i X_i \leq V$$

And

$$0 \leq X_i \leq Q_i.$$

## Example of a 0-1 KP

Suppose we have a knapsack that has a capacity of 13 cubic inches and several items of different sizes and different benefits. We want to include in the knapsack only these items that will have the greatest total benefit within the constraint of the knapsack's capacity. There are three potential items (labeled 'A,' 'B,' 'C'). Their volumes and benefits are as follows:

Item #	A	B	C
Benefit	4	3	5
Volume	6	7	8

We seek to maximize the total benefit:

$$\sum_{i=1}^3 B_i X_i = 4X_1 + 3X_2 + 5X_3$$

Subject to the constraints:

$$\sum_{i=1}^3 V_i X_i = 6X_1 + 7X_2 + 8X_3 \leq 13$$

And

$$X_i \in \{0,1\}, \text{ for } i= 1, 2, \dots, n.$$

For this problem there are  $2^3$  possible subsets of items:

<i>A</i>	<i>B</i>	<i>C</i>	<i>Volume of the set</i>	<i>Benefit of the set</i>
0	0	0	0	0
0	0	1	8	5
0	1	0	7	3
0	1	1	15	-
1	0	0	6	4
1	0	1	14	-
<i>1</i>	<i>1</i>	<i>0</i>	<i>13</i>	<i>7</i>
1	1	1	21	-

In order to find the best solution we have to identify a subset that meets the constraint and has the maximum total benefit. In our case, only rows given in italics satisfy the

## Implementation of the 0-1 KP Using GAs

### Representation of the items

We use a data structure, called *cell*, with two fields (benefit and volume) to represent every item. Then we use an array of type *cell* to store all items in it, which looks as follows:

items	0	1	2	3
	20   30	5   10	10   20	40   50

## Encoding of the chromosomes

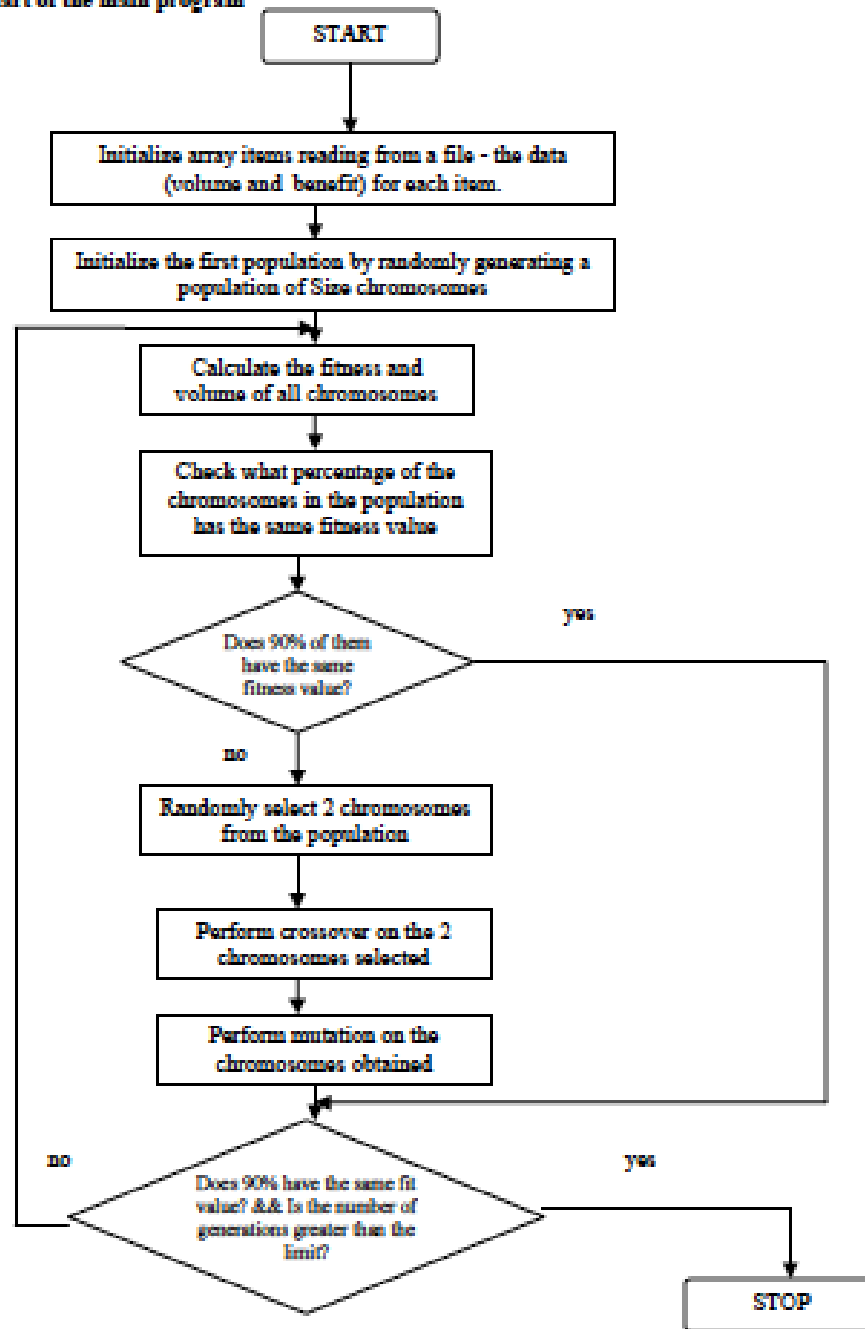
A chromosome can be represented in an array having size equal to the number of the items (in our example of size 4). Each element from this array denotes whether an item is included in the knapsack ('1') or not ('0'). For example, the following chromosome:

0	1	2	3
1	0	0	1

indicates that the 1<sup>st</sup> and the 4<sup>th</sup> item are included in the knapsack. To represent the whole population of chromosomes we use a tri-dimensional array (chromosomes [*Size*][*number of items*][2]). *Size* stands for the number of chromosomes in a population. The second dimension represents the number of items that may potentially be included in the knapsack. The third dimension is used to form the new generation of chromosomes.

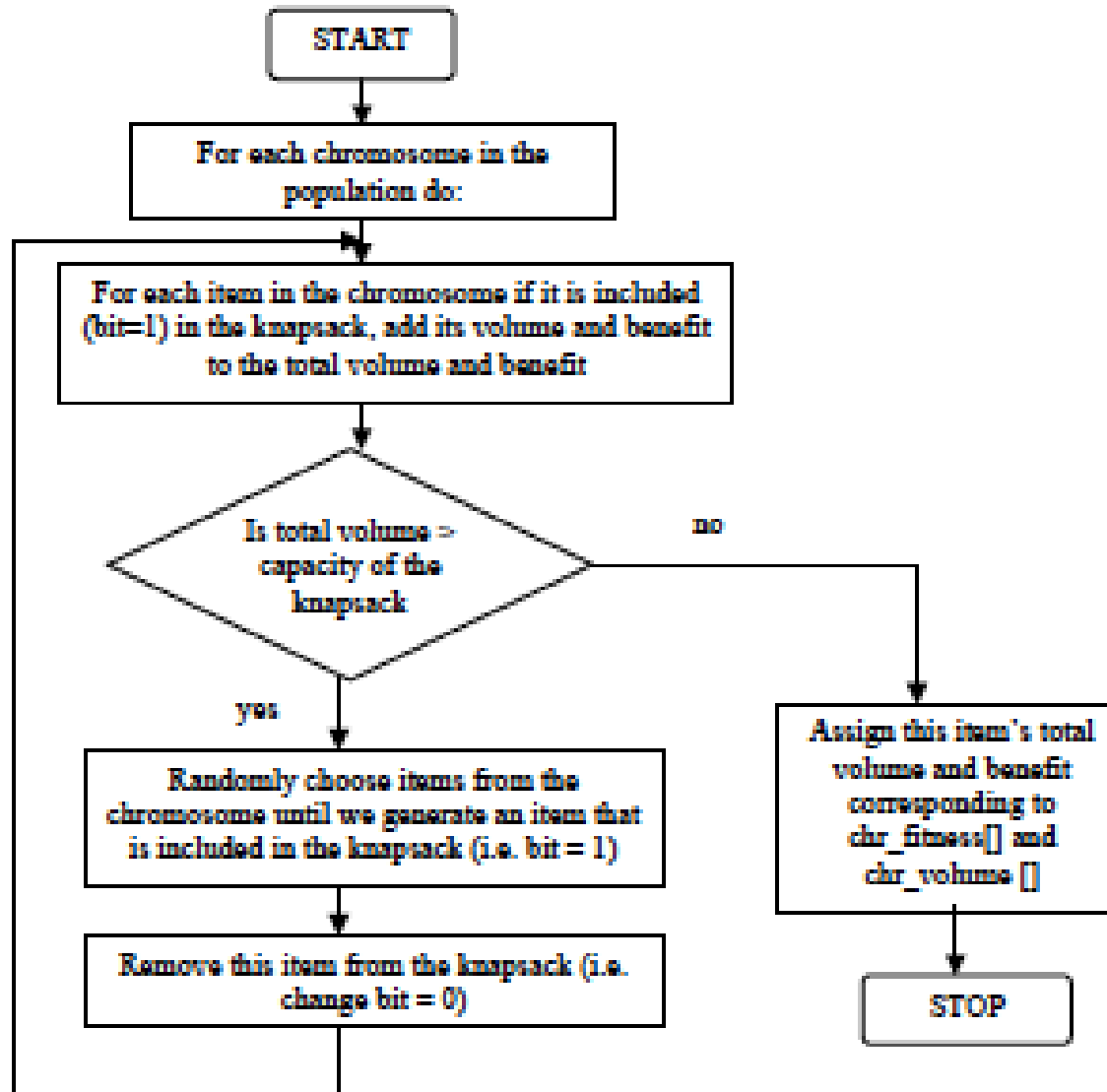


Flowchart of the main program



## Fitness function

We calculate the fitness of each chromosome by summing up the benefits of the items that are included in the knapsack, while making sure that the capacity of the knapsack is not exceeded. If the volume of the chromosome is greater than the capacity of the knapsack then one of the bits in the chromosome whose value is '1' is inverted and the chromosome is checked again. Here is a flowchart of the fitness function algorithm:

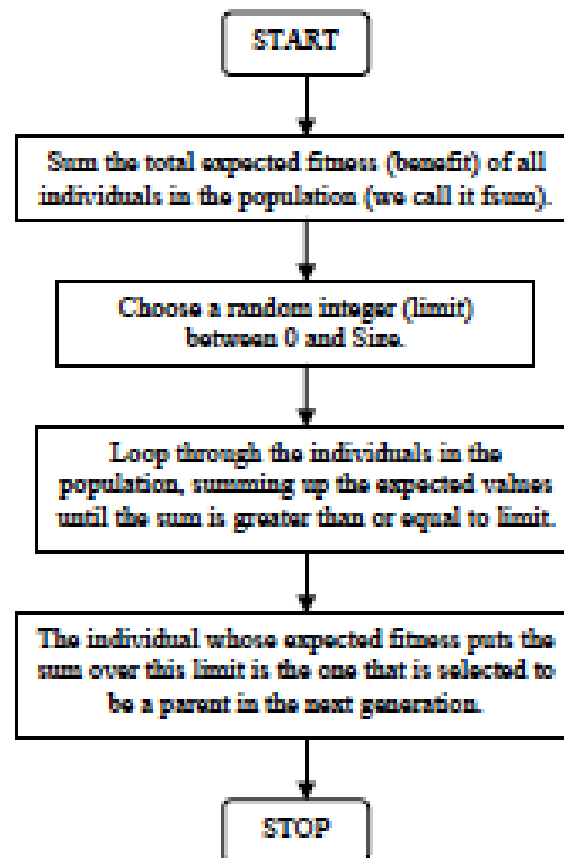


## Selection functions

In the implementation of the program, we tried two selection methods: roulette-wheel and group selection, combined with elitism, where two of the fittest chromosomes are copied without changes to the new population, so the best solutions found will not be lost.

### *Roulette-wheel selection*

Roulette-wheel is a simple method of implementing fitness-proportionate selection. It is conceptually equal to giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness [2]. The wheel is spun  $N$  times, where  $N$  is the number of the individuals in the population (in our case  $N = Size$ ). On each spin, the individual under wheel's marker is selected to be in the pool of parents for the next generation [2]. This method can be implemented in the following way:



### Group selection

We implemented this selection method in order to increase the probability of choosing fitter chromosomes to reproduce more often than chromosomes with lower fitness values. Since we could not find any selection method like this one in the literature, we decided to call it group selection.

For the implementation of the group selection method, we use another array *indexes[Size]*, where we put the indexes of the elements in the array *chr\_fitness[Size]*.

0	1	2	3	4	5	6	7	8	9	10	11
40	20	5	1	9	7	38	27	16	19	11	3

0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11

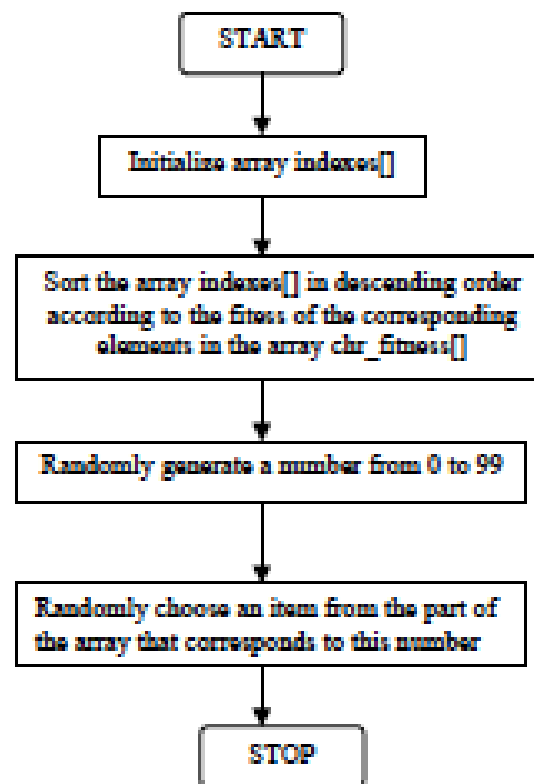
We sort the array *indexes* in descending order according to the fitness of the corresponding elements in the array *chr\_fitness*. Thus, the indexes of the chromosomes with higher fitness values will be at the beginning of the array *indexes*, and the ones with lower fitness will be towards the end of the array.

0	1	2	3	4	5	6	7	8	9	10	11
0	6	7	1	9	8	10	4	5	2	11	3

We divide the array into four groups:

- 0 ... 2 (0 ...  $\text{Size}/4$ )
- 3 ... 5 ( $\text{Size}/4$  ...  $\text{Size}/2$ )
- 6 ... 8 ( $\text{Size}/2$  ...  $3 \cdot \text{Size}/4$ )
- 9 ... 11 ( $3 \cdot \text{Size}/4$  ...  $\text{Size}$ )

Then, we randomly choose an item from the first group with 50% probability, from the second group with 30% probability, from the third group with 15% probability, and from the last group with 5% probability. Thus, the fitter a chromosome is the more chance it has to be chosen for a parent in the next generation. Here is a flowchart of the group selection algorithm:



*Comparison between the results from roulette-wheel and group selection methods*

Crossover Ratio = 85%

Mutation Ratio = 0.1%

No Elitism

Population Size	Roulette-wheel selection			Group selection method		
	Nb of Gen	Max. Fit. found	Items chosen	Nb of Gen	Max. Fit. found	Items chosen
100	62	2310	2, 5, 8, 13, 15	39	3825	1, 2, 3, 4, 5, 7, 9, 12
200	75	2825	1, 2, 6, 7, 8, 17	51	4310	1, 2, 3, 4, 5, 6, 7, 8, 11
300	91	2825	2, 3, 5, 7, 8, 16	53	4315	1, 2, 3, 4, 5, 6, 7, 8, 10
400	59	2825	1, 2, 3, 4, 15, 16	49	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
500	64	2835	1, 3, 6, 8, 10, 11	65	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
750	97	2840	3, 4, 5, 7, 9, 10	45	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
1000	139	2860	1, 3, 4, 6, 8, 12	53	4320	1, 2, 3, 4, 5, 6, 7, 8, 9

Crossover Ratio = 85%

Mutation Ratio = 0.1%

Elitism - two of the fittest chromosomes are copied without changes to a new population

Population Size	Roulette-wheel selection			Group selection method		
	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>
100	60	3845	1, 2, 3, 4, 5, 7, 8, 9	42	3840	1, 2, 3, 4, 5, 6, 7, 12
200	36	3830	1, 2, 3, 5, 6, 7, 8, 10	45	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
250	45	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	45	4315	1, 2, 3, 4, 5, 6, 7, 8, 10
300	46	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	27	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
400	43	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	47	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
500	25	4310	1, 2, 3, 4, 5, 6, 7, 9, 10	54	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
750	34	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	49	4320	1, 2, 3, 4, 5, 6, 7, 8, 9

Crossover Ratio = 75%

Mutation Ratio = 0.1%

Elitism - two of the fittest chromosomes are copied without changes to a new population

Population Size	Roulette-wheel selection			Group selection method		
	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>
100	49	3840	1, 2, 3, 4, 6, 7, 8, 9	46	3840	1, 2, 3, 4, 5, 7, 8, 10
200	42	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	43	4315	1, 2, 3, 4, 5, 6, 7, 8, 10
250	39	3820	1, 2, 3, 4, 6, 8, 9, 11	41	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
300	52	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	57	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
400	42	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	45	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
500	44	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	61	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
750	29	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	50	4320	1, 2, 3, 4, 5, 6, 7, 8, 9

Crossover Ratio = 95%

Mutation Ratio = 0.1%

Elitism - two of the fittest chromosomes are copied without changes to a new population

Population Size	Roulette-wheel selection			Group selection method		
	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>	<i>Nb of Gen</i>	<i>Max. Fit. found</i>	<i>Items chosen</i>
100	27	3320	1, 2, 3, 5, 9, 10, 13	37	3825	1, 2, 3, 4, 5, 6, 9, 13
200	49	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	50	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
250	62	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	40	4310	1, 2, 3, 4, 5, 6, 7, 9, 10

300	31	4320	1, 2, 3, 4, 5, 6, 7, 8, 9	60	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
400	38	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	50	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
500	25	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	56	4320	1, 2, 3, 4, 5, 6, 7, 8, 9
750	34	4315	1, 2, 3, 4, 5, 6, 7, 8, 10	54	4320	1, 2, 3, 4, 5, 6, 7, 8, 9

The results from the two selection functions, roulette-wheel and group selection, differ a lot depending on whether we used elitism to enhance their performance or not.

When we do not use elitism the group selection method is better than the roulette-wheel selection method, because with group selection the probability of choosing the best chromosome over the worst one is higher than it is with roulette-wheel selection method. Thus, the new generation will be formed by fitter chromosomes and have a bigger chance to approximate the optimal solution.

When we use elitism then the results from roulette-wheel and group selection method are similar because with elitism the two best solutions found throughout the run of the program will not be lost.

### Crossover

We tried both single and double point crossover. Since there was not a big difference in the results we got from both methods, we decided to use single point crossover. The crossover point is determined randomly by generating a random number between 0 and  $num\_items - 1$ . We perform crossover with a certain probability. If crossover probability is 100% then whole new generation is made by crossover. If it is 0% then whole new generation is made by exact copies of chromosomes from old population. We decided upon crossover rate of 85% by testing the program with different values. This means that 85% of the new generation will be formed with crossover and 15% will be copied to the new generation.

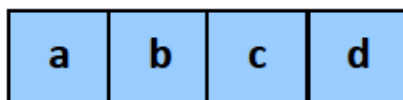
### Mutation

Mutation is made to prevent GAs from falling into a local extreme. We perform mutation on each bit position of the chromosome with 0.1 % probability.

## Numerical Example

Here are examples of applications that use genetic algorithms to solve the problem of combination. Suppose there is equality  $a + 2b + 3c + 4d = 30$ , genetic algorithm will be used to find the value of  $a$ ,  $b$ ,  $c$ , and  $d$  that satisfy the above equation. First we should formulate the objective function, for this problem

the objective is minimizing the value of function  $f(x)$  where  $f(x) = ((a + 2b + 3c + 4d) - 30)$ . Since there are four variables in the equation, namely  $a$ ,  $b$ ,  $c$ , and  $d$ , we can compose the chromosome as follow:



To speed up the computation, we can restrict that the values of variables  $a$ ,  $b$ ,  $c$ , and  $d$  are integers between 0 and 30.



## Step 1. Initialization

For example we define the number of chromosomes in population are 6, then we generate random of gene a, b, c, d for 6 chromosomes

**Chromosome**[1] = [a;b;c;d] = [12;05;23;08]

**Chromosome**[2] = [a;b;c;d] = [02;21;18;03]

**Chromosome**[3] = [a;b;c;d] = [10;04;13;14]

**Chromosome**[4] = [a;b;c;d] = [20;01;10;06]

**Chromosome**[5] = [a;b;c;d] = [01;04;13;19]

**Chromosome**[6] = [a;b;c;d] = [20;05;17;01]

## Step 2. Evaluation

We compute the objective function value for each chromosome produced in initialization step:

$$\begin{aligned}\mathbf{F\_obj}[1] &= \text{Abs}((12 + 2*05 + 3*23 + 4*08) - 30) \\ &= \text{Abs}((12 + 10 + 69 + 32) - 30) \\ &= \text{Abs}(123 - 30) \\ &= 93\end{aligned}$$

$$\begin{aligned}\mathbf{F\_obj}[2] &= \text{Abs}((02 + 2*21 + 3*18 + 4*03) - 30) \\ &= \text{Abs}((02 + 42 + 54 + 12) - 30) \\ &= \text{Abs}(110 - 30) \\ &= 80\end{aligned}$$

$$\begin{aligned}\mathbf{F\_obj}[3] &= \text{Abs}((10 + 2*04 + 3*13 + 4*14) - 30) \\ &= \text{Abs}((10 + 08 + 39 + 56) - 30) \\ &= \text{Abs}(113 - 30) \\ &= 83\end{aligned}$$

$$\begin{aligned}\mathbf{F\_obj}[4] &= \text{Abs}((20 + 2*01 + 3*10 + 4*06) - 30) \\ &= \text{Abs}((20 + 02 + 30 + 24) - 30) \\ &= \text{Abs}(76 - 30) \\ &= 46\end{aligned}$$

$$\begin{aligned}\mathbf{F\_obj}[5] &= \text{Abs}((01 + 2*04 + 3*13 + 4*19) - 30) \\ &= \text{Abs}((01 + 08 + 39 + 76) - 30) \\ &= \text{Abs}(124 - 30) \\ &= 94\end{aligned}$$

$$\begin{aligned}\mathbf{F\_obj}[6] &= \text{Abs}((20 + 2*05 + 3*17 + 4*01) - 30) \\ &= \text{Abs}((20 + 10 + 51 + 04) - 30) \\ &= \text{Abs}(85 - 30) \\ &= 55\end{aligned}$$

### Step 3. Selection

1. The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome. To avoid divide by zero problem, the value of  $F_{obj}$  is added by 1.

$$\begin{aligned}\text{Fitness}[1] &= 1 / (1 + F_{obj}[1]) \\ &= 1 / 94 \\ &= 0.0106\end{aligned}$$

$$\begin{aligned}\text{Fitness}[2] &= 1 / (1 + F_{obj}[2]) \\ &= 1 / 81 \\ &= 0.0123\end{aligned}$$

$$\begin{aligned}\text{Fitness}[3] &= 1 / (1 + F_{obj}[3]) \\ &= 1 / 84 \\ &= 0.0119\end{aligned}$$

$$\begin{aligned}
 \mathbf{Fitness}[4] &= 1 / (1 + \mathbf{F\_obj}[4]) \\
 &= 1 / 47 \\
 &= 0.0213
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{Fitness}[5] &= 1 / (1 + \mathbf{F\_obj}[5]) \\
 &= 1 / 95 \\
 &= 0.01 = 0.0845
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{Fitness}[6] &= 1 / (1 + \mathbf{F\_obj}[6]) \\
 &= 1 / 56 \\
 &= 0.0179
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{Total} &= 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179 \\
 &= 0.0845
 \end{aligned}$$

$$\begin{aligned}\mathbf{P}[1] &= 0.0106 / 0.0845 \\ &= 0.1254\end{aligned}$$

$$\begin{aligned}\mathbf{P}[2] &= 0.0123 / 0.0845 \\ &= 0.1456\end{aligned}$$

$$\begin{aligned}\mathbf{P}[3] &= 0.0119 / 0.0845 \\ &= 0.1408\end{aligned}$$

$$\begin{aligned}\mathbf{P}[4] &= 0.0213 / 0.0845 \\ &= 0.2521\end{aligned}$$

$$\begin{aligned}\mathbf{P}[5] &= 0.0105 / 0.0845 \\ &= 0.1243\end{aligned}$$

$$\begin{aligned}\mathbf{P}[6] &= 0.0179 / 0.0845 \\ &= 0.2118\end{aligned}$$

From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes. For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

$$C[1] = 0.1254$$

$$\begin{aligned} C[2] &= 0.1254 + 0.1456 \\ &= 0.2710 \end{aligned}$$

$$\begin{aligned} C[3] &= 0.1254 + 0.1456 + 0.1408 \\ &= 0.4118 \end{aligned}$$

$$\begin{aligned} C[4] &= 0.1254 + 0.1456 + 0.1408 + 0.2521 \\ &= 0.6639 \end{aligned}$$

$$\begin{aligned} C[5] &= 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 \\ &= 0.7882 \end{aligned}$$

$$\begin{aligned} C[6] &= 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118 \\ &= 1.0 \end{aligned}$$

Having calculated the cumulative probability of selection process using roulette-wheel can be done. The process is to generate random number  $R$  in the range 0-1 as follows.

$$R[1] = 0.201$$

$$R[2] = 0.284$$

$$R[3] = 0.099$$

$$R[4] = 0.822$$

$$R[5] = 0.398$$

$$R[6] = 0.501$$

If random number  $R$  [1] is greater than  $P$  [1] and smaller than  $P$  [2] then select Chromosome [2] as a chromosome in the new population for next generation:

$$\text{NewChromosome}[1] = \text{Chromosome}[2]$$

$$\text{NewChromosome}[2] = \text{Chromosome}[3]$$

$$\text{NewChromosome}[3] = \text{Chromosome}[1]$$

$$\text{NewChromosome}[4] = \text{Chromosome}[6]$$

$$\text{NewChromosome}[5] = \text{Chromosome}[3]$$

$$\text{NewChromosome}[6] = \text{Chromosome}[4]$$



Chromosome in the population thus became:

**Chromosome**[1] = [02;21;18;03]

**Chromosome**[2] = [10;04;13;14]

**Chromosome**[3] = [12;05;23;08]

**Chromosome**[4] = [20;05;17;01]

**Chromosome**[5] = [10;04;13;14]

**Chromosome**[6] = [20;01;10;06]

#### Step 4. Crossover

In this example, we use one-cut point, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome. Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using crossover\_rate (pc) parameters. Pseudo-code for the crossover process is as follows:

begin

$k \leftarrow 0$ ;

    while( $k < \text{population}$ ) do

$R[k] \leftarrow \text{random}(0-1)$ ;

        if ( $R[k] < \text{pc}$ ) then

            select **Chromosome**[k] as parent;

        end;

$k = k + 1$ ;

    end;

Chromosome  $k$  will be selected as a parent if  $R[k] < p_c$ . Suppose we set that the crossover rate is 25%, then Chromosome number  $k$  will be selected for crossover if random generated value for Chromosome  $k$  below 0.25. The process is as follows: First we generate a random number  $R$  as the number of population.

$$R[1] = 0.191$$

$$R[2] = 0.259$$

$$R[3] = 0.760$$

$$R[4] = 0.006$$

$$R[5] = 0.159$$

$$R[6] = 0.340$$

For random number R above, parents are Chromosome [1], Chromosome [4] and Chromosome [5] will be selected for crossover.

**Chromosome[1]  $\times$  Chromosome[4]**

**Chromosome[4]  $\times$  Chromosome[5]**

**Chromosome[5]  $\times$  Chromosome[1]**

After chromosome selection, the next process is determining the position of the crossover point. This is done by generating random numbers between 1 to (length of Chromosome – 1). In this case, generated random numbers should be between 1 and 3. After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged. For example we generated 3 random number and we get:

**C[1] = 1**

**C[2] = 1**

**C[3] = 2**

Then for first crossover, second crossover and third crossover, parent's gens will be cut at gen number 1, gen number 1 and gen number 3 respectively, e.g.

$$\begin{aligned}\mathbf{Chromosome}[1] &= \mathbf{Chromosome}[1] \times \mathbf{Chromosome}[4] \\ &= [02;21;18;03] \times [20;05;17;01] \\ &= [02;05;17;01]\end{aligned}$$

$$\begin{aligned}\mathbf{Chromosome}[4] &= \mathbf{Chromosome}[4] \times \mathbf{Chromosome}[5] \\ &= [20;05;17;01] \times [10;04;13;14] \\ &= [20;04;13;14]\end{aligned}$$

$$\begin{aligned}\mathbf{Chromosome}[5] &= \mathbf{Chromosome}[5] \times \mathbf{Chromosome}[1] \\ &= [10;04;13;14] \times [02;21;18;03] \\ &= [10;04;18;03]\end{aligned}$$

Thus Chromosome population after experiencing a crossover process:

**Chromosome**[1] = [02;05;17;01]

**Chromosome**[2] = [10;04;13;14]

**Chromosome**[3] = [12;05;23;08]

**Chromosome**[4] = [20;04;13;14]

**Chromosome**[5] = [10;04;18;03]

**Chromosome**[6] = [20;01;10;06]

## Step 5. Mutation

Number of chromosomes that have mutations in a population is determined by the mutation rate parameter. Mutation process is done by replacing the gen at random position with a new value. The process is as follows. First we must calculate the total length of gen in the population. In this case the total length of gen is  $\text{total\_gen} = \text{number\_of\_gen\_in\_Chromosome} * \text{number of population}$

$$= 4 * 6$$

$$= 24$$

Mutation process is done by generating a random integer between 1 and total\_gen (1 to 24). If generated random number is smaller than mutation\_rate(pm) variable then marked the position of gen in chromosomes. Suppose we define pm 10%, it is expected that 10% (0.1) of total\_gen in the population that will be mutated:

$$\text{number of mutations} = 0.1 * 24$$

$$= 2.4$$

$$\approx 2$$

Suppose generation of random number yield 12 and 18 then the chromosome which have mutation are Chromosome number 3 gen number 4 and Chromosome 5 gen number 2. The value of mutated gens at mutation point is replaced by random number between 0-30. Suppose generated random number are 2 and 5 then Chromosome composition after mutation are:

- Chromosome[1] = [02;05;17;01]
- Chromosome[2] = [10;04;13;14]
- Chromosome[3] = [12;05;23;**02**]
- Chromosome[4] = [20;04;13;14]
- Chromosome[5] = [10;**05**;18;03]
- Chromosome[6] = [20;01;10;06]



Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:

**Chromosome**[1] = [02;05;17;01]

$$\begin{aligned}\mathbf{F\_obj}[1] &= \text{Abs}((02 + 2*05 + 3*17 + 4*01) - 30) \\ &= \text{Abs}((2 + 10 + 51 + 4) - 30) \\ &= \text{Abs}(67 - 30) \\ &= 37\end{aligned}$$

**Chromosome**[2] = [10;04;13;14]

$$\begin{aligned}\mathbf{F\_obj}[2] &= \text{Abs}((10 + 2*04 + 3*13 + 4*14) - 30) \\ &= \text{Abs}((10 + 8 + 33 + 56) - 30) \\ &= \text{Abs}(107 - 30) \\ &= 77\end{aligned}$$

**Chromosome[3] = [12;05;23;02]**

$$\begin{aligned}\mathbf{F\_obj[3]} &= \text{Abs}((12 + 2*05 + 3*23 + 4*02) - 30) \\ &= \text{Abs}((12 + 10 + 69 + 8) - 30) \\ &= \text{Abs}(87 - 30) \\ &= 47\end{aligned}$$

**Chromosome[4] = [20;04;13;14]**

$$\begin{aligned}\mathbf{F\_obj[4]} &= \text{Abs}((20 + 2*04 + 3*13 + 4*14) - 30) \\ &= \text{Abs}((20 + 8 + 39 + 56) - 30) \\ &= \text{Abs}(123 - 30) \\ &= 93\end{aligned}$$

**Chromosome[5] = [10;05;18;03]**

$$\begin{aligned}\mathbf{F\_obj[5]} &= \text{Abs}((10 + 2*05 + 3*18 + 4*03) - 30) \\ &= \text{Abs}((10 + 10 + 54 + 12) - 30) \\ &= \text{Abs}(86 - 30) \\ &= 56\end{aligned}$$

**Chromosome[6] = [20;01;10;06]**

$$\begin{aligned}\mathbf{F\_obj[6]} &= \text{Abs}((20 + 2*01 + 3*10 + 4*06) - 30) \\ &= \text{Abs}((20 + 2 + 30 + 24) - 30) \\ &= \text{Abs}(76 - 30) = 46\end{aligned}$$

From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous Chromosome generation. New Chromosomes for next iteration are:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;02]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;05;18;03]

Chromosome[6] = [20;01;10;06]

These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration. This process will be repeated until a predetermined number of generations. For this example, after running 50 generations, best chromosome is obtained:

Chromosome = [07; 05; 03; 01]

This means that:

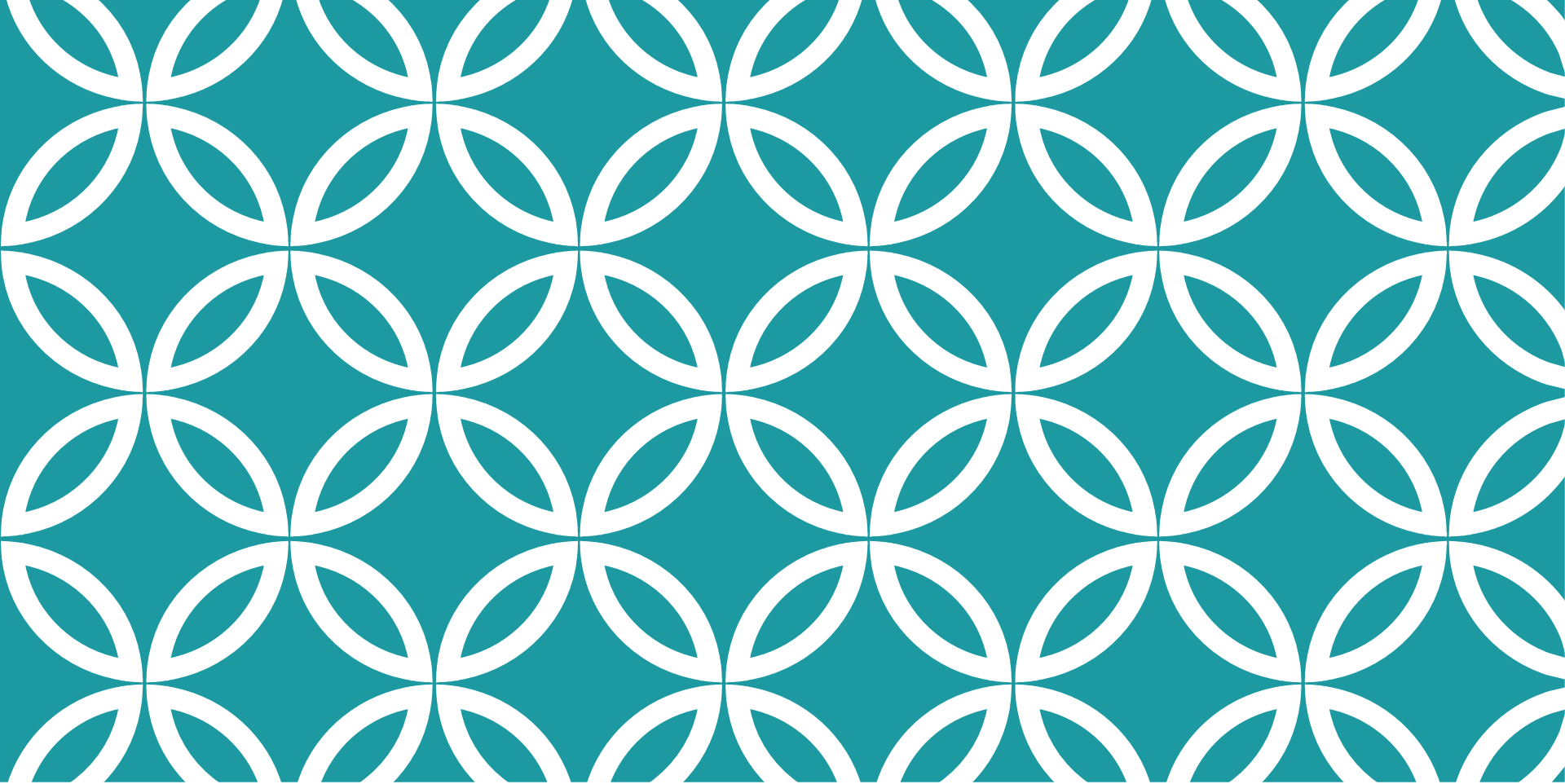
$$a = 7, b = 5, c = 3, d = 1$$

If we use the number in the problem equation

$$a + 2b + 3c + 4d = 30$$

$$7 + (2 * 5) + (3 * 3) + (4 * 1) = 30$$

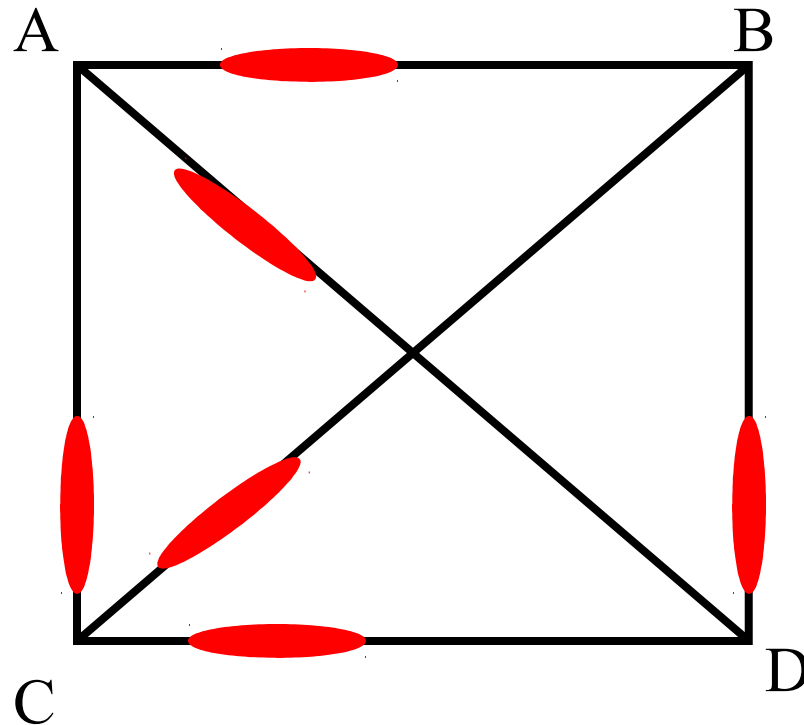
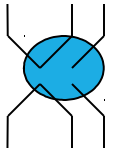
We can see that the value of variable a, b, c and d generated by genetic algorithm can satisfy that equality.



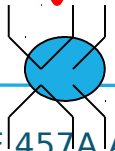
# ANT COLONY

# E.G. A 4-CITY TSP

Initially, random levels of pheromone are scattered on the edges



Pheromone

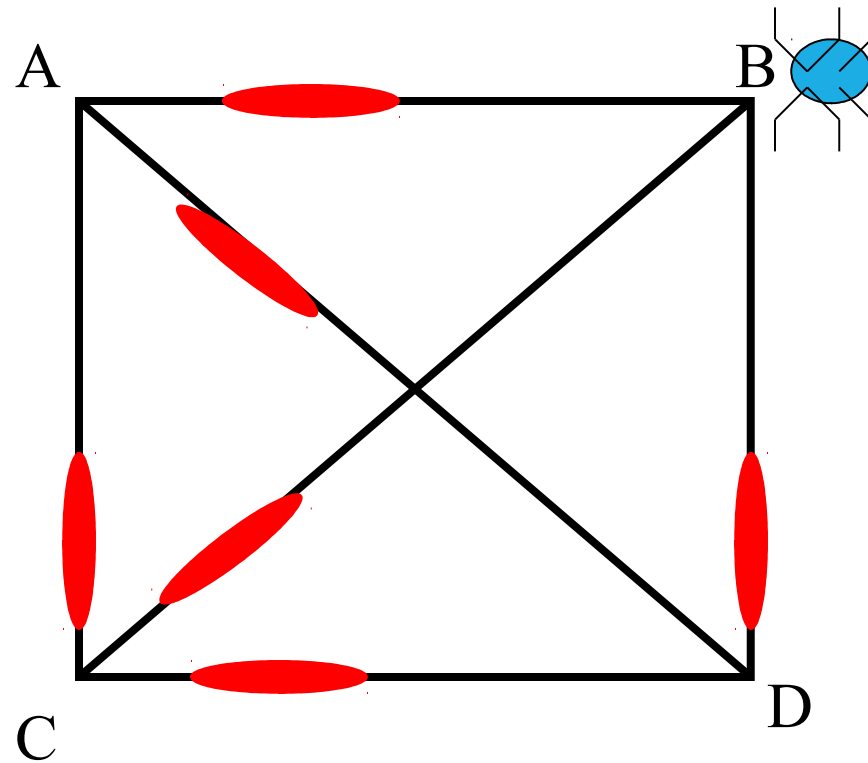


Ant

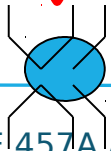
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

# E.G. A 4-CITY TSP

An ant is placed at a random node



Pheromone



Ant

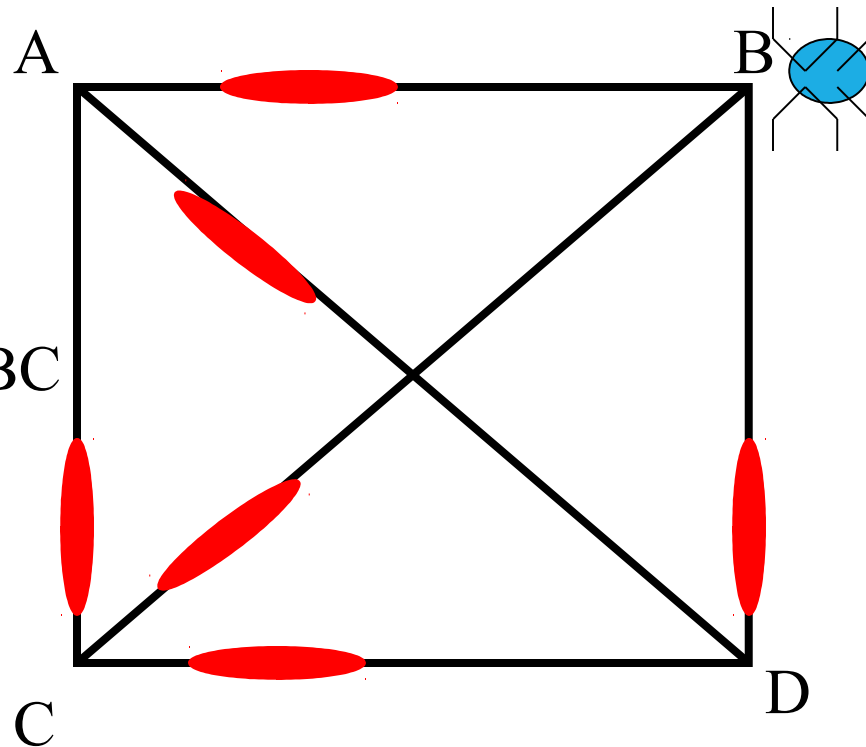
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

# E.G. A 4-CITY TSP

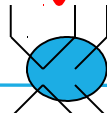
The ant decides where to go from that node, based on probabilities calculated from:

- pheromone strengths,
- next-hop distances.

Suppose this one chooses BC



 Pheromone

 Ant

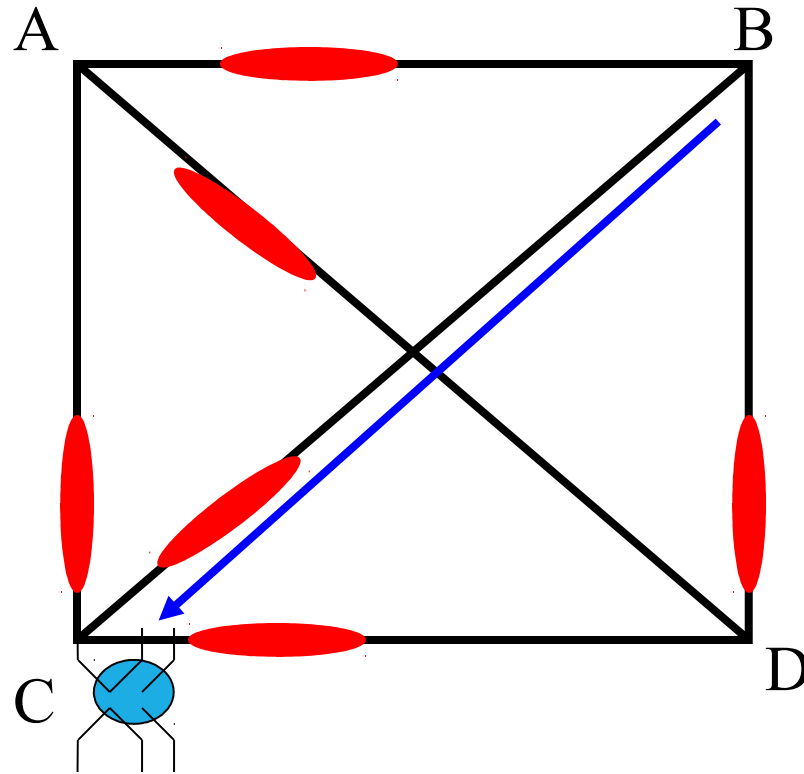
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20



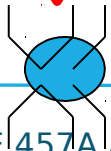
# E.G. A 4-CITY TSP

The ant is now at C, and has a 'tour memory' = {B, C} – so he cannot visit B or C again.

Again, he decides next hop (from those allowed) based on pheromone strength and distance; suppose he chooses CD



Pheromone

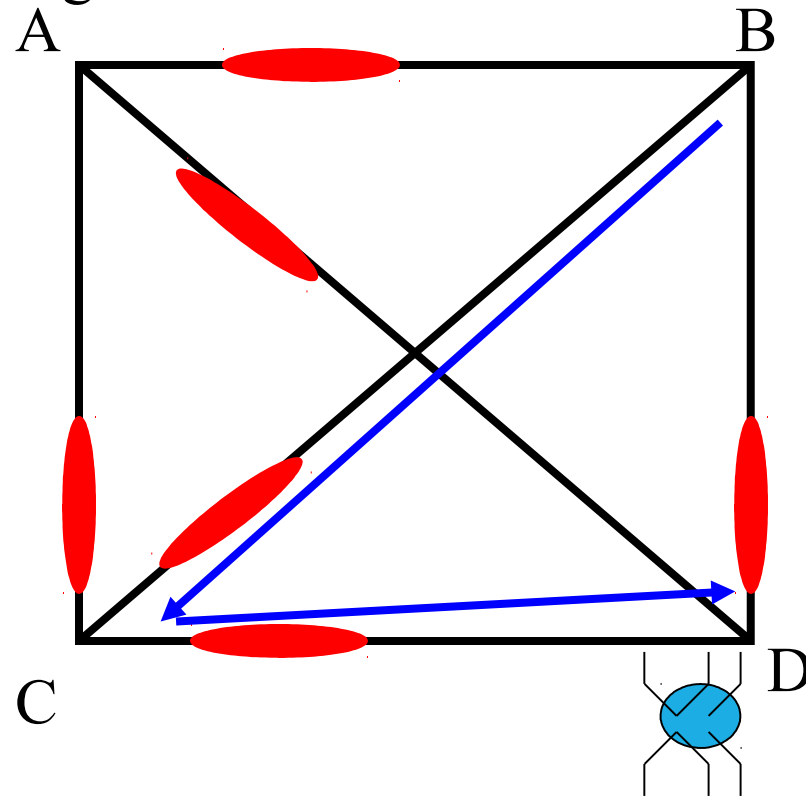


Ant

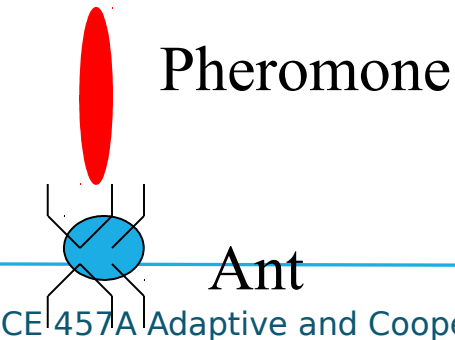
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

# E.G. A 4-CITY TSP

The ant is now at D, and has a 'tour memory' = {B, C, D}  
There is only one place he can go now:

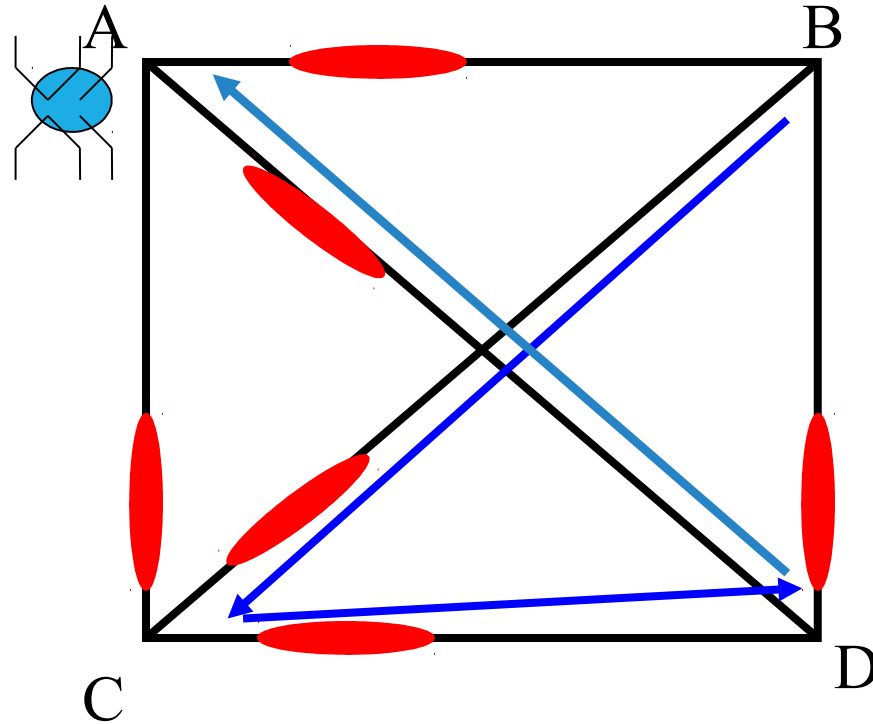


AB: 10, AC: 10, AD: 30, BC: 40, CD: 20



# E.G. A 4-CITY TSP

So, he has nearly finished his tour, having gone over the links:  
BC, CD, and DA.



Pheromone

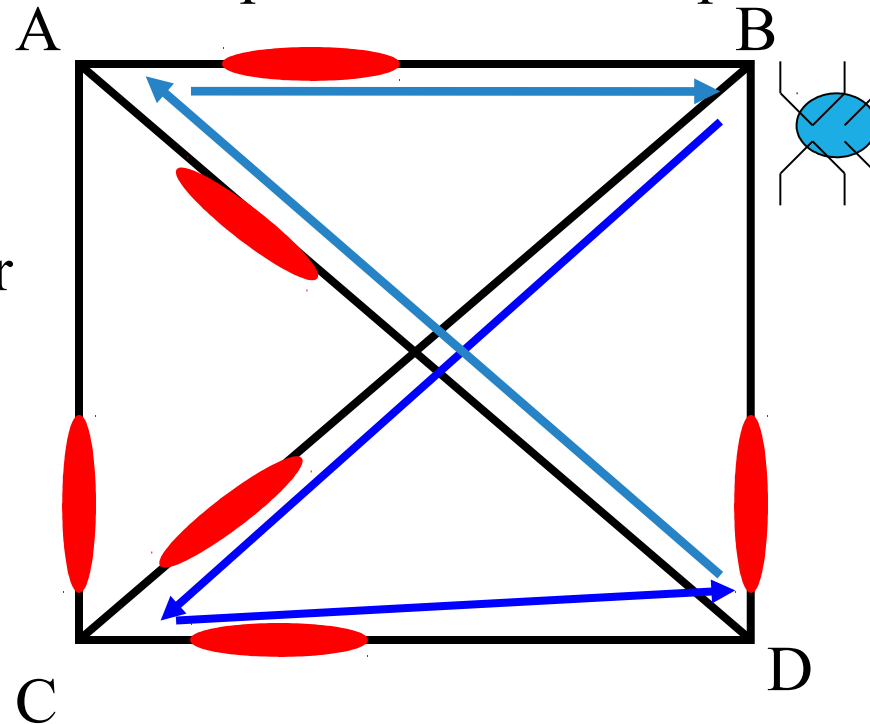
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

Ant

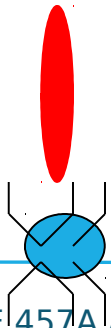
# E.G. A 4-CITY TSP

So, he has nearly finished his tour, having gone over the links: BC, CD, and DA. AB is added to complete the round trip.

Now, pheromone on the tour is increased, in line with the fitness of that tour.



Pheromone

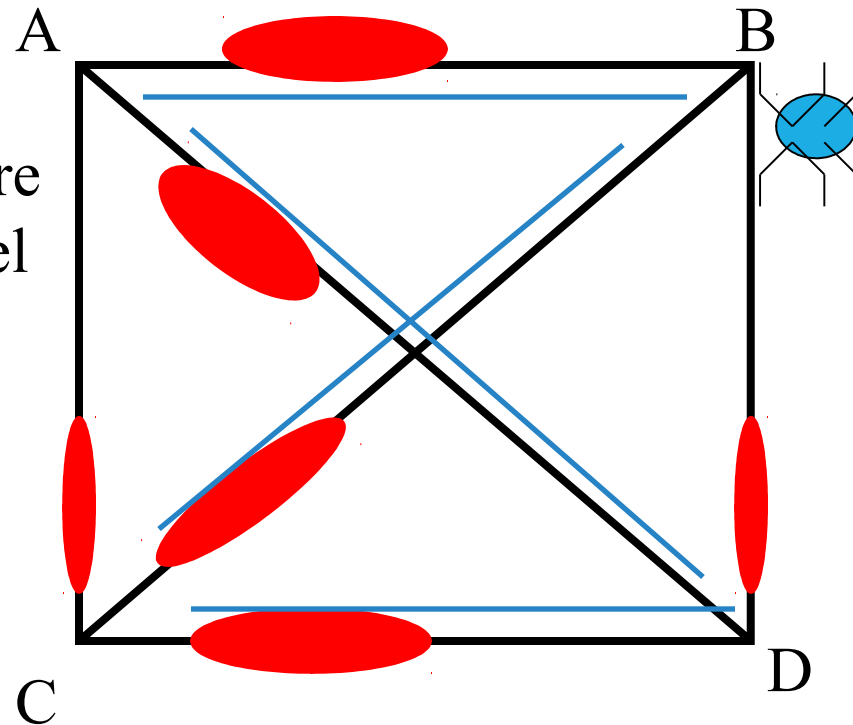


Ant

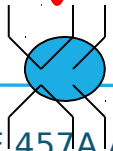
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

# E.G. A 4-CITY TSP

Next, pheromone everywhere is decreased a little, to model decay of trail strength over time



Pheromone



Ant

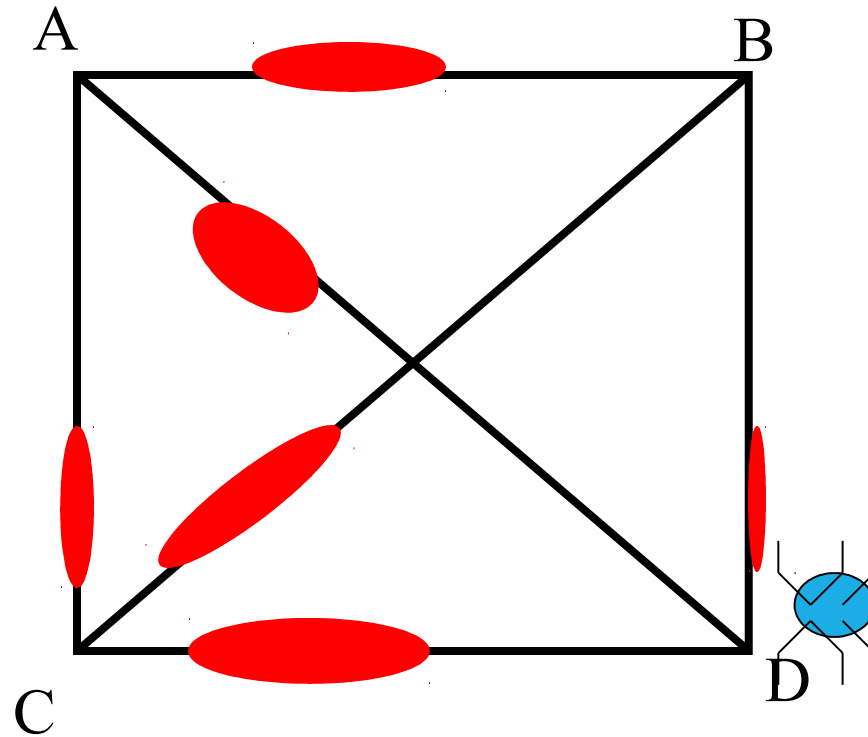
AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

# E.G. A 4-CITY TSP

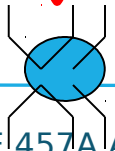
We start again, with another ant in a random position.

Where will he go?

Next, the actual algorithm and variants.



Pheromone



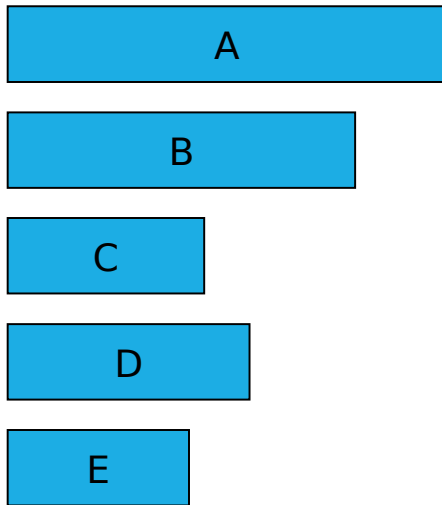
Ant

AB: 10, AC: 10, AD: 30, BC: 40, CD: 20

E.G.

# SINGLE MACHINE SCHEDULING WITH DUE-DATES

These jobs have to be done; their length represents the time they will take.

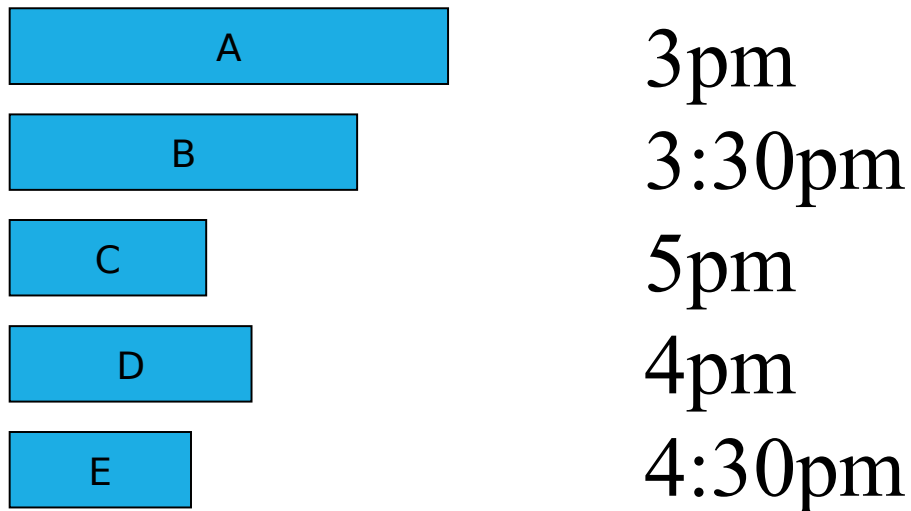


# E.G.

## SINGLE MACHINE SCHEDULING WITH DUE-DATES

These jobs have to be done; their length represents the time they will take.

Each has a 'due date', when it needs to be finished



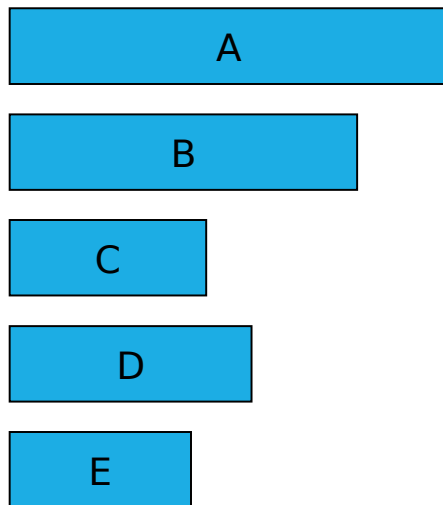


# E.G.

## SINGLE MACHINE SCHEDULING WITH DUE-DATES

These jobs have to be done; their length represents the time they will take.

Each has a 'due date', when it needs to be finished



3pm

3:30pm

5pm

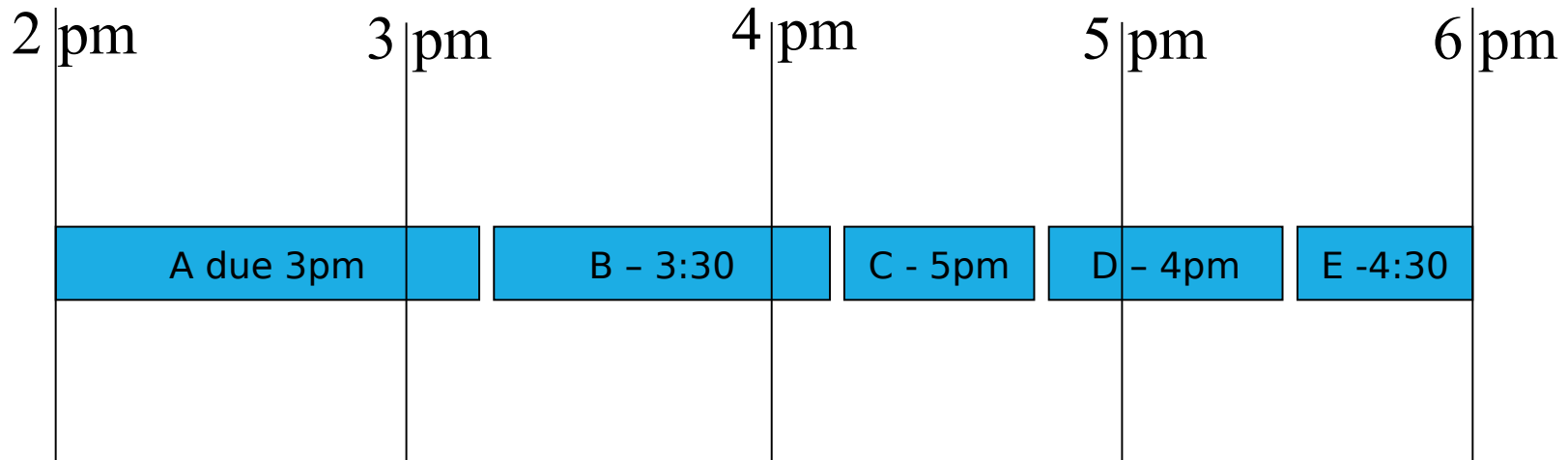
4pm

4:30pm

Only one 'machine' is available to process these jobs, so can do just one at a time.

[e.g. machine might be human tailor, photocopier, Hubble Space Telescope, Etc ...]

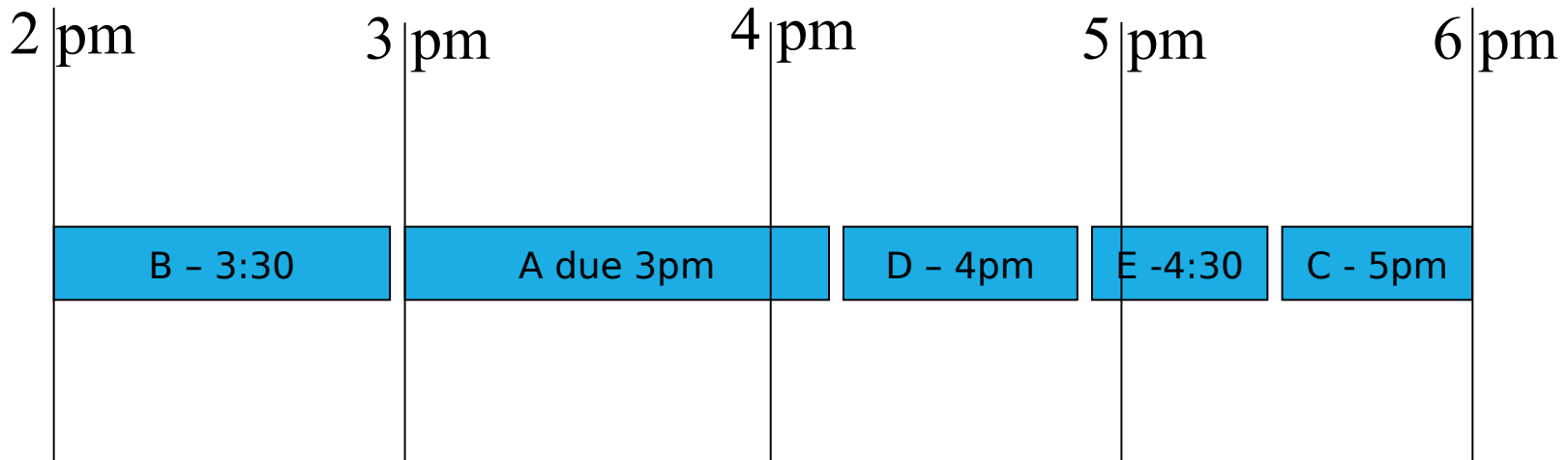
# AN EXAMPLE SCHEDULE



A is 10min late  
B is 40min late  
C is 20min early (lateness = 0)  
D is 90min late  
E is 90min late

Fitness might be average lateness;  
in this case 46min  
or fitness could be Max lateness,  
in this case 90min

# ANOTHER SCHEDULE



A is 70min late  
B is 30min early (0 lateness)  
C is 60min late  
D is 50min late  
E is 50min late

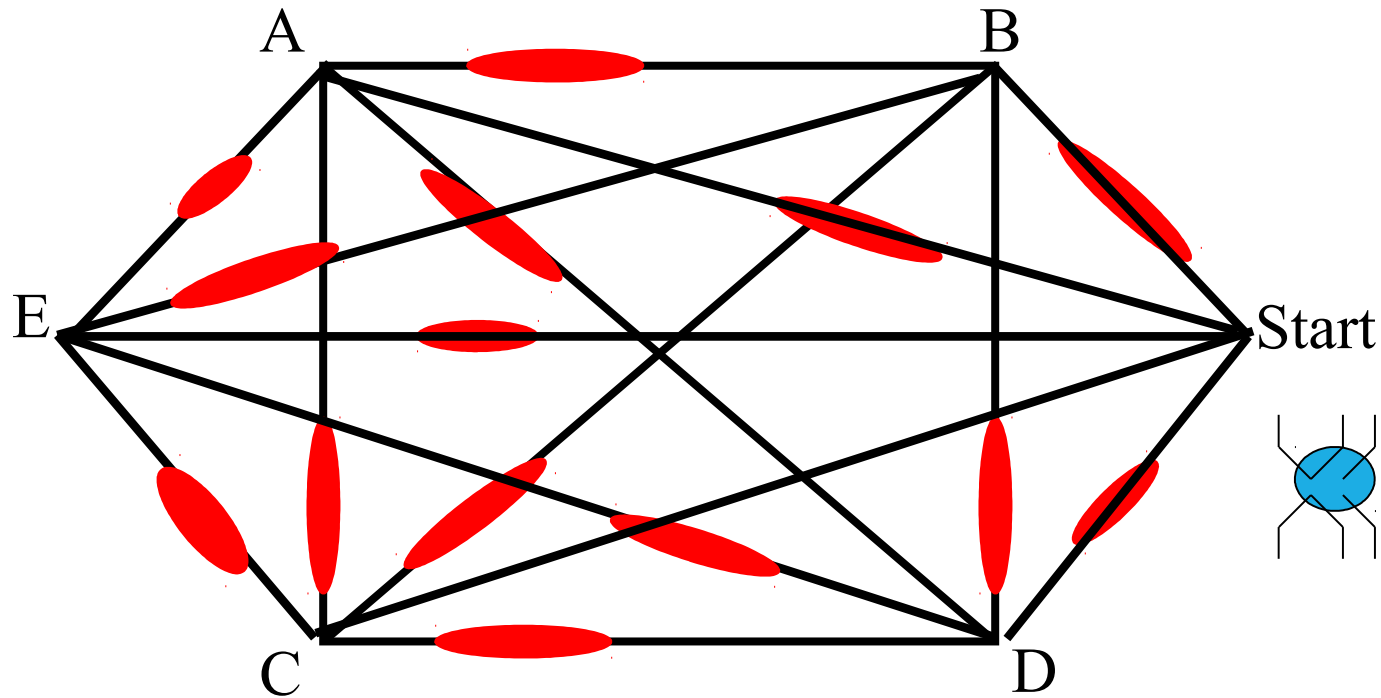
Fitness might be average lateness;  
in this case again 46min

or fitness could be Max lateness,  
in this case 70min

# APPLYING ACO TO THIS PROBLEM

Just like with the TSP, each ant finds paths in a network, where, in this case, each job is a node. Also, no need to return to start node – path is complete when every node is visited.

Initially, random levels of pheromone are scattered on the edges, an ant starts at a *Start* node (so the first link it chooses defines the first task to schedule on the machine); as before it uses a transition rule to take one step at a time, biased by pheromone levels, and also a heuristic score, each time choosing the next machine to schedule. What heuristic might you use in this case?



Why is it sensible to have a Start node for this problem

# PSO TYPICAL QUESTIONS

- I. Write the basic PSO algorithm,
- II. Provide equations for the position and velocity update of particles,
- III. Discuss different neighborhood topologies and their impact on the update equations,
- IV. Discuss how to use PSO for permutation problems and use the traveling sales person as an example to illustrate your answer,
- V. Describe the TRIBES method for adapting the PSO to make it parameter free.

# I. PSO – A SIMPLE ALGORITHM (SYNCHRONOUS UPDATE)

Initialize the swarm,

While *termination criteria* is not met

- ▢ For each particle
  - ▢ Update the particle's velocity,
  - ▢ Update the particle's position,
  - ▢ Update the particle's personal best,
- end for

- ▢ Update the Nbest,

end while

# PSO – A DIFFERENT ALGORITHM (ASYNCHRONOUS UPDATE)

Initialize the swarm,

While *termination criteria* is not met

▢ For each particle

- ▢ Update the particle's velocity,
- ▢ Update the particle's position,
- ▢ Update the particle's personal best,
- ▢ Update the Nbest,

end for

end while



The neighbourhood best  
update is moved into the  
particles update loop



## II. PSO

Equations of motion:

$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id} (pbest_t^{id} - x_t^{id}) + c_2 r_2^{id} (Nbest_t^{id} - x_t^{id})$$
$$x_{t+1}^{id} = x_t^{id} + v_{t+1}^{id}$$

where

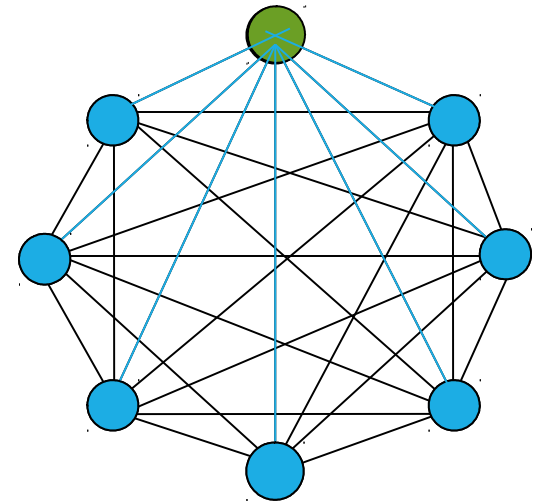
- $w$  is the inertia weight,
- $c_1, c_2$  are the acceleration coefficients,
- $r_1, r_2$  are randomly generated numbers in  $[0, 1]$ ,
- $t$  is the iteration number,
- $i$  and  $d$  are the particle number and the dimension.

# III. PSO - NEIGHBOURHOODS

*gbest model:* each particle is influenced by all the other particles,

The fastest Propagation of information in a population,

Particles can get stuck easily in local minima.

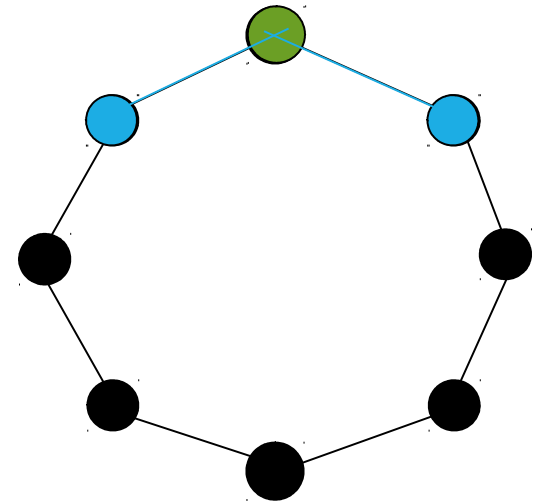


# PSO - NEIGHBOURHOODS

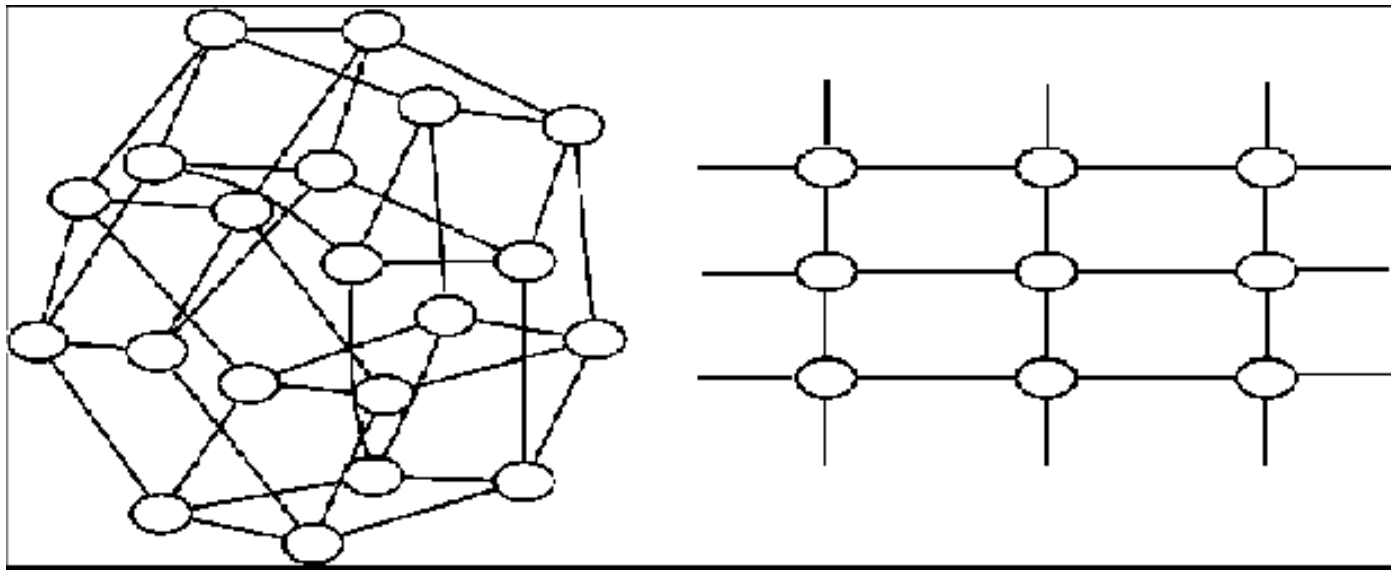
*lbest model:* each particle is influenced only by particles in its own neighbourhood,

The propagation of information is the slowest,

Doesn't get stuck easily in local minima but might increase the computational cost.



# PSO - NEIGHBOURHOODS



The complete population

A local region

## **The Von Neumann model [10]**


## IV. PSO FOR TSP

Here we need to use a method to map from the continuous space of PSO to the permutation space.

One approach is to define

The position of a particle as the solution to a problem (permutation of cities),

And the velocity of a particle as the set of swaps to be performed on a particle

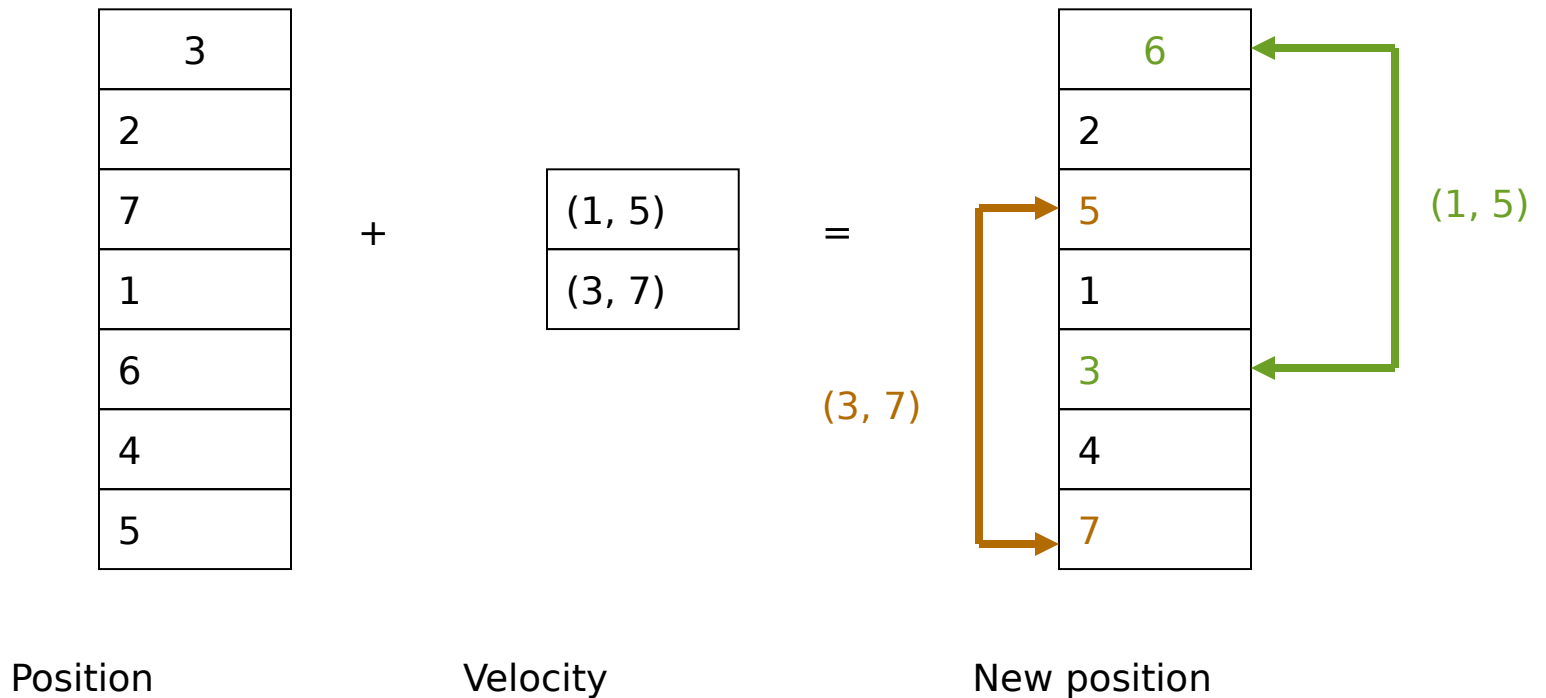


Three operations were re-defined for the new search space:

- ▮ Adding a velocity to a position,
- ▮ Subtracting two positions,
- ▮ Multiplying a velocity by a constant.

# EXAMPLE

Adding a velocity to a position:



Another approach is to use continuous PSO

In order to evaluate the particle fitness, they performed a *space transformation* from a particle in the continuous domain to a permutation in the solution space.

all the elements in the position vector (along with their indices) sorted to get a sorted list in descending order,

The sorted indices are taken as the permutation.



Example:

*if*  $x = [0.2, 0.3, 0.1, 0.8]$   
then the sorted list would be:

$$\begin{array}{cccc} & 4 & 2 & 1 & 3 \\ x_{sorted} = & [0.8, 0.3, 0.2, 0.1] \end{array}$$
  
and the permutation would be:

Tour to be evaluated and its  $\longrightarrow P = [4, 2, 1, 3]$

cost is the particles' x fitness

## V. ADAPTATION

An adaptive parameter free PSO, referred to as *TRIBES*, was proposed in [17],

The purpose was to have the user to call PSO algorithm without the need to set any parameters,

The main point was to adapt the number particles used in the search.

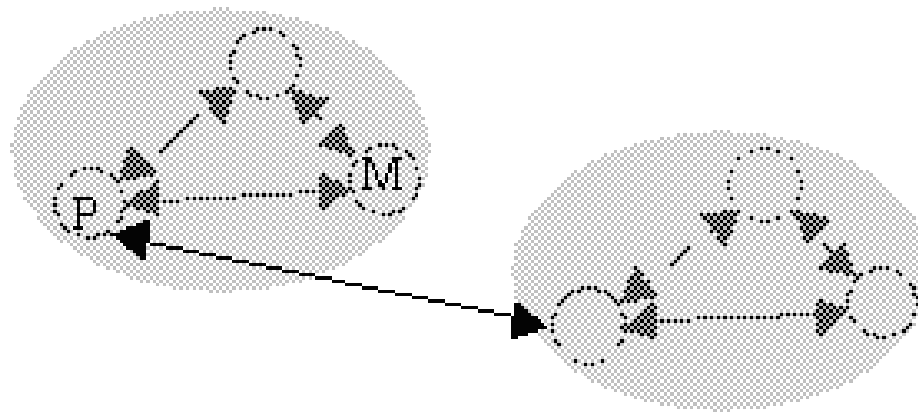
# ADAPTATION

A *tribe* refers to a group of connected particles,

All the tribes should have some type of connection between them to inform one another of their findings,

This will help in deciding which is the global minimum among all the different solutions that was found by the different tribes.

# ADAPTATION



A connection  
between two tribes

# ADAPTATION

## Definitions:

- ▢ A *good* particle is a particle that has its pbest improved in the last iteration, otherwise it's *neutral*,
- ▢ Each particle memorizes the last two performance variations, a particle with both variations as improvements is an *excellent* particle,
- ▢  $G$  is the number of good particles in a tribe.

# ADAPTATION

Definitions:

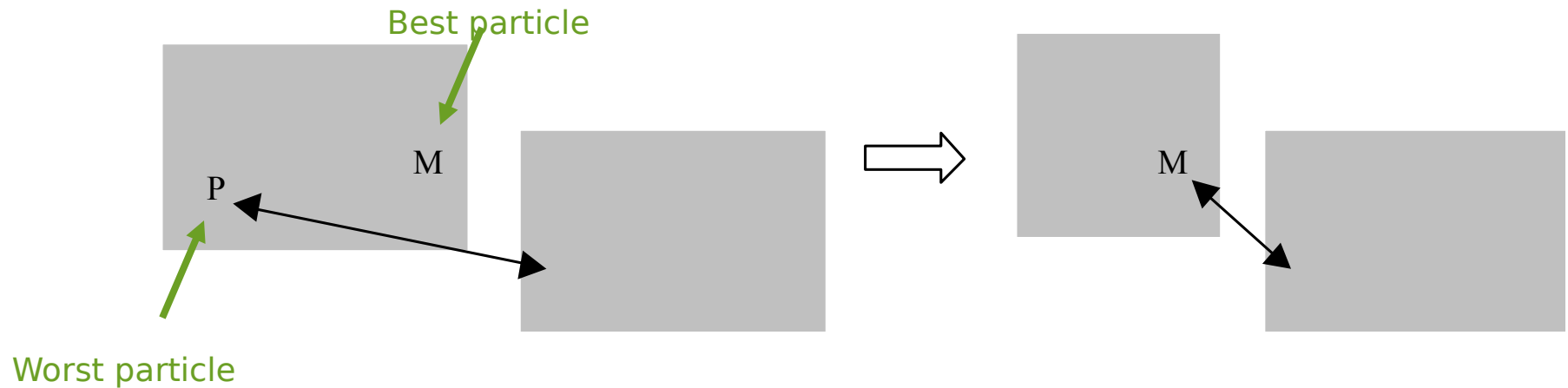
- A tribe is marked as good depending on the value of  $G$ , if the total number of particles in a tribe is  $T$  :

$$r = U(0, 1)$$

$$Tribe = \begin{cases} \text{good}, & r < \frac{G}{T} \\ \text{bad}, & \text{otherwise} \end{cases}$$

# ADAPTATION

A good tribe deletes its worst particle to conserve the number of performed function evaluations.



# ADAPTATION

On the other hand, every bad tribe generates a new random particle simultaneously,

All the new particles form a new tribe,

Each particle gets connected to the tribe that generated it through its best particle.



# ADAPTATION

The idea is to start with a single particle,

Most likely, this particle won't improve in the first iteration. Hence, it will generate another particle forming another tribe,

If both don't improve, they will simultaneously generate two other particles forming a third two-particle tribe,

And the process continues...

# ADAPTATION

If things go bad, larger and larger tribes will be generated to increase the swarm search power,

On the other hand, if good solutions start to occur, good tribes will start removing its worst particles reducing the tribes size possibly to complete extinction.

# GENETIC PROGRAMMING

# GENETIC

# PROGRAMMING

Specialized form of GA

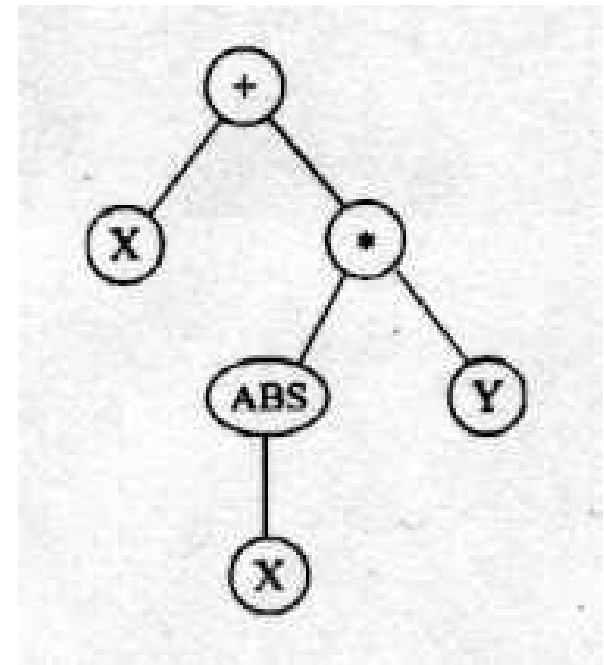
Manipulates a very **specific type** of solution using modified genetic operators

Original application was to design **computer programs**

Now applied in alternative areas eg. **Analog Circuits**

Does not make distinction between search and solution space.

Solution represented in very specific **hierarchical** manner.



# BACKGROUND/HISTORY

By John R. Koza, Stanford University.

1992, Genetic Programming Treatise -

“Genetic Programming. On the Programming of Computers by Means of Natural Selection.”

- Origin of GP.

**Combining the idea** of machine learning and evolved tree structures.

# WHY GENETIC PROGRAMMING?

It saves time by **freeing the human** from having to design complex algorithms.

Not only designing the algorithms but creating ones that give **optimal** solutions.

Again, Artificial Intelligence.

# WHAT CONSTITUTES A GENETIC PROGRAM?

Starts with "What needs to be done"

Agent figures out "How to do it"

Produces a computer program - "Breeding Programs"

Fitness Test

Code reuse

Architecture Design - Hierarchies

Produce results that are competitive with human produced results

# HOW ARE GENETIC PRINCIPLES APPLIED?

“Breeding” computer programs.

Crossovers.

Mutations.

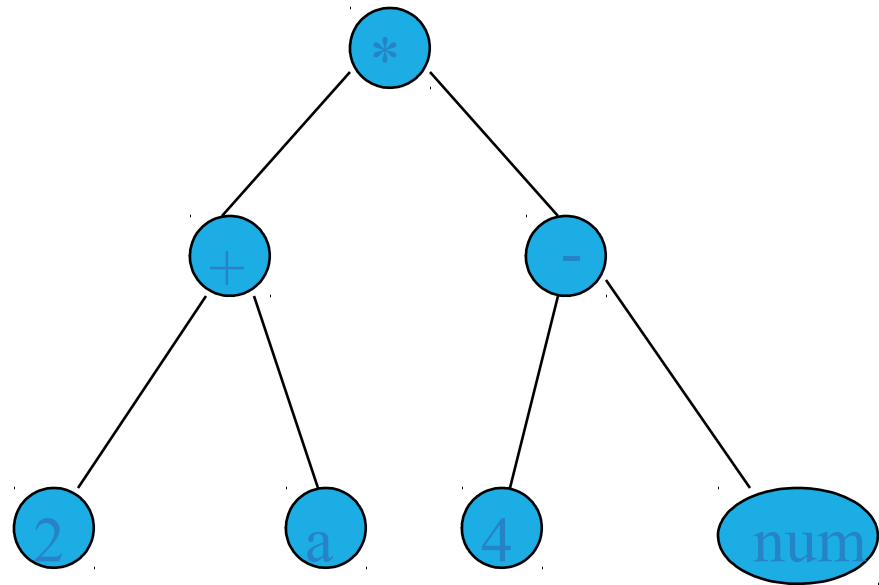
Fitness testing.



# COMPUTER PROGRAMS AS TREES

Infix/Postfix

$(2 + a) * (4 - \text{num})$



# “BREEDING” COMPUTER PROGRAMS



Hmm hmm heh.  
Hey butthead. Do  
computer programs  
actually score?

# “BREEDING” COMPUTER PROGRAMS

Start off with a large “pool” of random computer programs.

Need a way of coming up with the best solution to the problem using the programs in the “pool”

Based on the definition of the problem and criteria specified in the fitness test, mutations and crossovers are used to come up with new programs which will solve the problem.

# THE FITNESS TEST

# THE FITNESS TEST

Identifying the way of evaluating how good a given computer program is at solving the problem at hand.

How good can a program cope with its **environment**.

Can be measured in many ways, i.e. **error**, **distance**, **time**, etc...

# FITNESS TEST CRITERIA

Time complexity a good criteria.

□ i.e.  $n^2$  vs.  $n \log n$ .

□ “How long to find the solution?”

Accuracy - Values of variables.

□ “How good is the solution?”

Combinations of criteria may also be tested.

# MUTATIONS

)

# MUTATIONS IN NATURE

## Properties of mutations

Ultimate source of genetic variation.

**Radiation, chemicals** change genetic information.

Causes new genes to be created.

One chromosome.

Asexual.

Very rare.

Before:

acgtactggctaa

**After:**

acatactggctaa



# GENETIC PROGRAMMING

**1.** Randomly generate a combinatorial set of **computer programs**.

**2.** Perform the following steps iteratively until a termination criterion is satisfied

- ▢ a. Execute each program and assign a fitness value to each individual.
- ▢ b. **Create a new population** with the following steps:
  - ▢ i. **Reproduction**: Copy the selected program unchanged to the new population.
  - ▢ ii. **Crossover**: Create a new program by recombining two selected programs at a random crossover point.
  - ▢ iii. **Mutation**: Create a new program by randomly changing a selected program.

**3.** The best sets of individuals are deemed the optimal solution upon termination

# MUTATIONS IN PROGRAMS

Single parental program is ***probabilistically selected*** from the population based on fitness.

**Mutation** point randomly chosen.

- ▢ the subtree rooted at that point is deleted, and
- ▢ a **new subtree is grown** there using the same random growth process that was used to generate the initial population.

**Asexual operations** (mutation) are typically performed sparingly:

- ▢ with a *low probability* of mutations,
- ▢ probabilistically selected from the population based on fitness.

# CROSSTOVERS IN PROGRAMS

# CROSSTOVERS IN PROGRAMS

1. Two parental programs are selected from the population based on fitness.
2. A crossover point is randomly chosen in the first and second parent.
  1. The first parent is called **receiving**
  2. The second parent is called **contributing**
3. The **subtree** rooted at the crossover point of the first parent is deleted
4. It is replaced by the subtree from the second parent.
5. **Crossover** is the predominant operation in **genetic programming** (and **genetic algorithm**) research
6. It is performed with a high probability (say, 85% to 90%).

# GP

Assume you are asked to solve a regression analysis problem that you do not know the exact property of the function. Regression analysis tries to estimate the dependency among variables. In order to do that the values from data is compared to values found by the estimation function. In least square minimization, the square of the differences are minimized. Mathematically you have the collected data  $z = d(x, y)$  for a set of  $x, y$  values and you are asked to estimate  $z = f(x, y)$  in order to minimize  $\sum_{\forall x, y} (f(x, y) - d(x, y))^2$

You know that the function can be represented by combination of addition, subtraction, multiplication, *sin*, *ln*, *exp* ( $e^x$ ). Also constants in the range  $[-100, 100]$  take part.

1. If you are asked to implement a tree encoded Genetic Programming solution to this problem, what will be your terminals and non-terminals.
2. Assume you need up to 4 digits after the decimal dot for constants. How many bits do you need to represent each constant.

	ES	EP	GA	GP
<b>Representation</b>	Real-valued	Real-valued	Binary-Valued	Lisp S-expressions
<b>Self-Adaptation</b>	Standard deviations and covariances	Variance	None	None
<b>Fitness</b>	Objective function values	Scaled objective function value	Scaled objective function value	Scaled objective function value
<b>Mutation</b>	Main operator	Only operator	Background operator	Background operator
<b>Recombination</b>	Different variants, important for self-adaptation	None	Main Operator	Main Operator
<b>Selection</b>	Deterministic extinctive	Probabilistic, extinctive	Probabilistic, preservative	Probabilistic, preservative

# TYPICAL QUESTIONS

Explain the difference between a genotypic representation and a phenotypic representation. Give an example of each.

Outline the similarities and differences between Genetic Algorithms and Evolutionary Strategies.