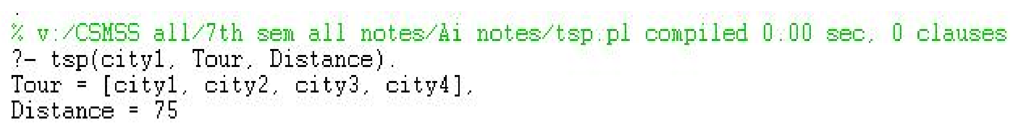
Traveling Salesman Problem with Brute Force:

Program:

```
% Define cities and distances between them.
city_distance(city1, city2, 10).
city_distance(city1, city3, 15).
city_distance(city1, city4, 20).
city_distance(city2, city3, 35).
city_distance(city2, city4, 25).
city_distance(city3, city4, 30).

% Entry point for the TSP solver.
tsp(StartCity, OptimalTour, MinDistance) :-
    findall(City, city_distance(StartCity, City, _), Cities),
    permutation(Cities, Permutation),
    calculate_distance([StartCity | Permutation], Distance),
    MinDistance is Distance,
    OptimalTour = [StartCity | Permutation].

% Calculate the total distance of a tour.
calculate_distance([_], 0).
calculate_distance([City1, City2 | Rest], TotalDistance) :-
    city_distance(City1, City2, Distance),
    calculate_distance([City2 | Rest], RemainingDistance),
    TotalDistance is Distance + RemainingDistance.

% Example query
% Replace 'city1' with the starting city of your choice.
% ?- tsp(city1, Tour, Distance).
```

Output:



```
% v:/CSMSS all/7th sem all notes/Ai notes/tsp.pl compiled 0.00 sec, 0 clauses
?- tsp(city1, Tour, Distance).
Tour = [city1, city2, city3, city4],
Distance = 75
```

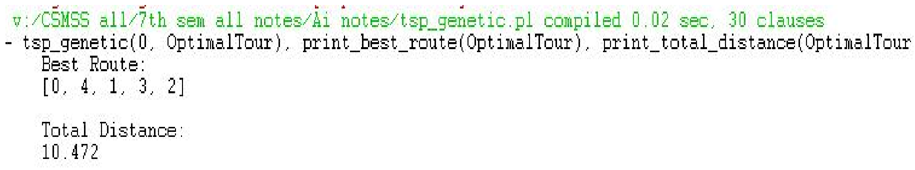
Traveling Salesman Problem with Genetic Algorithm:
Program:

```
:- use_module(library(random)).

% Define the number of cities
num_cities(5).

% Define the population size for the GA
population_size(10).

% Define the mutation rate for the GA
mutation_rate(0.1).

% Define cities
city(0, 0).
city(1, 2).
city(3, 1).
city(4, 3).
city(2, 4).

% Generate a random route
generate_random_route(Route) :-
    num_cities(NumCities),
    length(Route, NumCities),
    numlist(0, NumCitiesMinusOne, Cities),
    random_permutation(Cities, Route).

% Calculate the total distance of a route
calculate_total_distance(Route, TotalDistance) :-
    append(Route, [Route[0]], ClosedRoute), % Close the route
    calculate_total_distance_helper(ClosedRoute, TotalDistance).

calculate_total_distance_helper([City1, City2 | Rest], TotalDistance) :-
    city(City1, X1-Y1),
    city(City2, X2-Y2),
    DX is X1 - X2,
    DY is Y1 - Y2,
    Distance is sqrt(DX*DX + DY*DY),
    calculate_total_distance_helper([City2 | Rest], RestDistance),
    TotalDistance is Distance + RestDistance.
calculate_total_distance_helper([_], 0).

% Perform crossover between two parent routes to produce a child route
crossover(Parent1, Parent2, Child) :-
    length(Parent1, Length),
    random_between(1, Length, CrossoverPoint),
    append(Prefix, Suffix, Parent1),
    append(Prefix, RestParent2, Parent2),
    append(RestParent2, Suffix, Child).

% Mutate a route by swapping two cities
mutate(Route, MutatedRoute) :-
    mutation_rate(MutationRate),
    (random_float < MutationRate ->
        random_permutation(Route, MutatedRoute)
```

```prolog
    ;     MutatedRoute = Route
    ).

% Create an initial population
initialize_population(StartCity, Population) :-
    population_size(PopSize),
    findall(Route, (between(1, PopSize, _), generate_initial_route(StartCity, Route)), Population).

generate_initial_route(StartCity, Route) :-
    num_cities(NumCities),
    length(Route, NumCities),
    numlist(0, NumCitiesMinusOne, Cities),
    random_permutation(Cities, ShuffledCities),
    select(StartCity, ShuffledCities, Route).

% Evolutionary algorithm iteration
evolve_population([], []).
evolve_population([Parent1, Parent2 | Rest], [Child1, Child2 | NewPopulation]) :-
    crossover(Parent1, Parent2, Child1),
    crossover(Parent2, Parent1, Child2),
    mutate(Child1, MutatedChild1),
    mutate(Child2, MutatedChild2),
    evolve_population(Rest, NewPopulation).

% Perform the GA iterations
ga_iteration(Population, NewPopulation) :-
    evolve_population(Population, Children),
    append(Population, Children, CombinedPopulation),
    sort_population(CombinedPopulation, SortedPopulation),
    take_best(SortedPopulation, PopulationSize, NewPopulation).

% Sort the population based on fitness (total distance)
sort_population(Population, SortedPopulation) :-
    predsort(compare_fitness, Population, SortedPopulation).

compare_fitness(Order, Route1, Route2) :-
    calculate_total_distance(Route1, Fitness1),
    calculate_total_distance(Route2, Fitness2),
    compare(Order, Fitness1, Fitness2).

% Take the best N individuals from the population
take_best([], _, []).
take_best(Population, N, BestPopulation) :-
    length(Population, Length),
    MaxN is min(N, Length),
    take_best_helper(Population, MaxN, BestPopulation).

take_best_helper(_, 0, []).
take_best_helper([Individual | Rest], N, [Individual | BestRest]) :-
    N > 0,
    N1 is N - 1,
    take_best_helper(Rest, N1, BestRest).

% Main function
tsp_genetic(StartCity, OptimalTour) :-
    initialize_population(StartCity, Population),
    ga_iterations(Population, max_generations, OptimalTour).
```

```
% Perform GA iterations
ga_iterations(Population, 0, BestRoute) :-
        find_best_route(Population, BestRoute).
ga_iterations(Population, GenerationsLeft, BestRoute) :-
        ga_iteration(Population, NewPopulation),
        NextGenerationsLeft is GenerationsLeft - 1,
        ga_iterations(NewPopulation, NextGenerationsLeft, BestRoute).

% Print the best route
print_best_route(BestRoute) :-
        writeln('Best Route:'),
        writeln(BestRoute).

% Print the total distance of the best route
print_total_distance(BestRoute) :-
        calculate_total_distance(BestRoute, Fitness),
        writeln('Total Distance:'),
        writeln(Fitness).
```

Output:



```
 v:/CSMSS all/7th sem all notes/Ai notes/tsp_genetic.pl compiled 0.02 sec, 30 clauses
- tsp_genetic(0, OptimalTour), print_best_route(OptimalTour), print_total_distance(OptimalTour
   Best Route:
   [0, 4, 1, 3, 2]

   Total Distance:
   10.472
```