

The Physical Traveling Salesperson Problem *

A mixture of techniques

Jose Antonio Martin H.
Instituto de Automática Industrial, -C.S.I.C.-
jamartin@iai.csic.es

ABSTRACT

A mixture of techniques is presented in order to solve The Physical Traveling Salesperson Problem for GECCO-2005 Competition. The method of solution is divided in two principal parts. The first part is to find a good tour using the Classic Travel Salesman problem with a Cross Entropy algorithm and the second part is an heterogeneous mixture of distinct evolutionary algorithms that are specially designed and tuned for this problem.

Categories and Subject Descriptors: I.2.8 [Computing Methodologies][Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms: Algorithms, Theory

Keywords:Traveling Salesperson Problem , TSP, Physical Traveling Salesperson Problem, PTSP, Differential Evolution, Pattern Search, Gradient Evolution, Genetic Algorithms

1. INTRODUCTION

A mixture of techniques is presented in order to solve The Physical Traveling Salesperson Problem for GECCO-2005 Competition. The method of solution is divided in two principal parts. The first part is to find a good tour using the Classic Travel Salesman Problem with a Cross Entropy algorithm.

The second part is a mixture of heterogeneous distinct evolutionary algorithms that are specially designed and tuned for this problem. The huge dimensionality of the problem makes it in practice unaffordable for brute force method and classical branch a bound methods, so we have programmed a kind of robot that must visit all the cities, the brain of this robot is then parameterized in order to evolve best brains with evolutionary algorithms and thus obtain best solutions to the problem addressed.

2. FINDING A GOOD TOUR

The First step of the optimization process was to find a good tour, that is, a good order to visit all the cities. The approach that we have taken was to solve a simple TSP problem with the particularity that the traveling agent does

*The author wish to thank, Enrique Muñoz (Kike), Jesus Reviejo , R.Haber and Javier Alonso for their collaboration.

not have to complete a circuit. In order to do that, we have used a Cross Entropy algorithm with a modification of the cost matrix. The cost matrix will no be symmetrical.

We have added an additional city that represent the starting point of the PTSP problem and we have established the cost matrix by setting the cost to reach the starting point to 0, that is, the cost to travel from every city to the starting point is 0, this simple modification let us to use an existing traveling salesman problem solver without any modification.

We have compared these results with a simulated annealing algorithm for the TSP implemented by R. Haber, using the length of a spline curve as the cost function, this modification of the cost function produces tours that are in practice identical to the others approach.

3. DESIGN THE PARAMETERIZED ROBOT

In order to obtain at least a first feasible approximate solution, a robot that visit all the cities was made. This robot was programmed with the next formulation:

First, it is possible to derive the explicit analytic formula of the Newton acceleration and replacing some constants in order to obtain an expression for the acceleration, hence, here we have not used this approach because this yields to some suboptimal solutions due to the discrete control variables, so, we have opted for controlling the velocity instead of the acceleration.

Thus, the expression for obtaining the change in the velocity vector is:

$$\vec{X} = (X + \vec{V}.dt) \quad (1)$$

where X is the actual position of the robot, \vec{V} is the actual velocity and dt a differential time step, so the measure \vec{X} will be the predicted position of the robot.

Now for calculating a measure of the robot velocity correction we have used the Equation 2.

$$dX = T_x - X_x \quad dY = T_y - X_y \quad (2)$$

where T is the next target coordinate and $(\overrightarrow{dX, dY})$ is the predicted difference in the position of the robot and the next target location.

Next, due to the problem structure, there is a little trouble: We can only update the velocity of the robot with one vector at a time, so, we need to decide which component of the velocity is more important to update at each time step. This point is one of the most crucial parts of the robot design. We have opted here for the most obvious one, calculate

the tangent of the angle between the target location and the current predicted location. This calculus is quite simple and is computed in the next equation:

$$ax = \text{abs}(dY)/\text{abs}(dX) \quad ay = \text{abs}(dX)/\text{abs}(dY) \quad (3)$$

Thus,

```

if ax>=1
    if dY>0 out = 3
    if dY<0 out = 1

if ay>=1
    if dX>0 out = 2
    if dX<0 out = 4

```

In order to interpret these *out* values, see the Physical Traveling Salesman Problem Contest.

With this suboptimal solution but at least feasible solution we can evolve best controllers for the robot in order to obtain more and more better solutions to the problem. But, how can we evolve such model?.

The approximation selected for doing that is composed by simple rules:

1. If there is a constant somewhere then replace it by a parameter.
2. The next target location is not strict because there is a radius of 5, so, add a parameter for each component of the target location for each target.
3. The *dt* factor is explicitly a parameter of the model.
4. Between to target location we can add a sub target as a parameter.

So, with these simple rules, we can reformulate the robot model and make a parameterized robot that can produce more optimal solutions.

The complete parameterized formulation of the robot can be seen in the next equations:

We will denote p_i where $i = [1..n]$ for the parameters of the model.

$$\vec{X} = (X + \vec{V} \cdot p_1 \cdot p_2) \quad (4)$$

We have added two parameters to represent the differential time step *dt*.

$$dX = T_x + p_3 - X_x \quad dY = T_y + p_4 - X_y \quad (5)$$

We have added two parameters, one for each component of the next target.

$$ax = \text{abs}(dY)/\text{abs}(dX) \quad ay = \text{abs}(dX)/\text{abs}(dY) \quad (6)$$

```

if ax>=p5
    if dY>0 out = 3
    if dY<0 out = 1

if ay>=p6
    if dX>0 out = 2
    if dX<0 out = 4

```

3.1 Implementation Details

Given de parameters in the past section, we have clear which will be the genotype vector. a vector of 8 parameters for each city.

This way of parametrization allow us the use of a specialized set of parameters for each section between two cities, so, in total, the genotype will be a vector of $n \times nparams$ where n is the number of cities in the Map and $nparams$ is the number of parameters of the function that must return the next force value.

Once we have selected the genotype, we only need to select which evolutionary strategy we will follow. We are working with MatLab Genetic Algorithm Toolbox, so we have only two types of evolutionary algorithms, Pattern Search and a 'tradicional GA'. We are using these algorithms in a cycle, so the process of optimization and re-optimization never ends.

The communication between these algorithms is quite simple, once a Pattern Search returns the best solution, we need to create an initial population for the GA, for doing so, we add at the first; the best solution returned by Pattern Search and the rest of the population is the same individual plus a normal distributed individuals around the best.

In an infinite cycle we have four distinct evolutionary algorithms.

1. Standard GA with real parameters and Gaussian Mutation.
2. Pattern Search (MatLab).
3. A Standard GA that approximate the Gradient of the objective function.
4. A differential evolution implementation.

3.1.1 Step by Step Optimization

Another variant that was implemented for the final Map was the Step by Step Interactive Optimization, that is, optimizing the Global Map, segment by segment saving the final position and velocity of the segment and then optimizing the next segment starting with the saved position and velocity of the last segment optimized, this technique improves the optimization a lot, and give us the opportunity of being competitive.

3.1.2 The Code of the GetForce Function

The most important thing for our method is the function that returns the force value to be applied at each time step, so we have decided to publish here its code in MatLab code.

The parameters of this function are:

- x: The robot actual position
- target: The robot next target location
- vel: the robot actual velocity
- p: a vector of parameters from the part of the genotype that corresponds to this target.

```
function [out] = evgetac( x , target, vel, p)

warning off MATLAB:divideByZero

t = target;

a dynamic point between the next target
and the current location

ptarget = ( (t + x) / (2) ) .* [p(7) p(8)];

dist = sum((x-t).^2);
rz = dist / sum((x-ptarget).^2) ;

t = ( t*rz + (ptarget) ) / ( rz+1);

fx = x + ( vel * p(1) * p(6) ) .* [p(9) p(10)];
```

```
dX = ((t(1) + p(4) ) - fx(1) );
dY = ((t(2) + p(5) ) - fx(2) );
```

```
out = 0;
ax = abs(dY)/abs(dX);
ay = abs(dX)/abs(dY);
```

```
if ax>p(2)
    if dY>0
        out = 3;
        return
    end;
    if dY<0
        out = 1;
        return
    end;
end;
```

```
if ay>p(3)
    if dX>0
        out = 2;
        return
    end;
    if dX<0
        out = 4;
        return
    end;
end;
```

```
end;
end
```

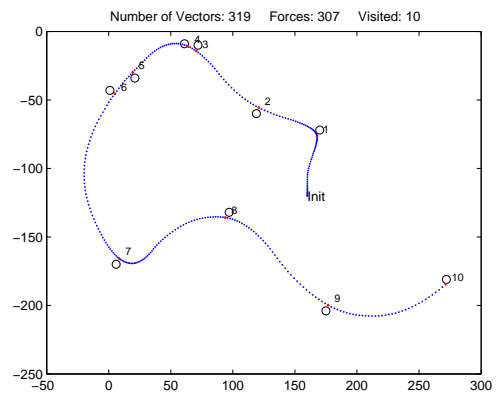


Figure 1: Map101

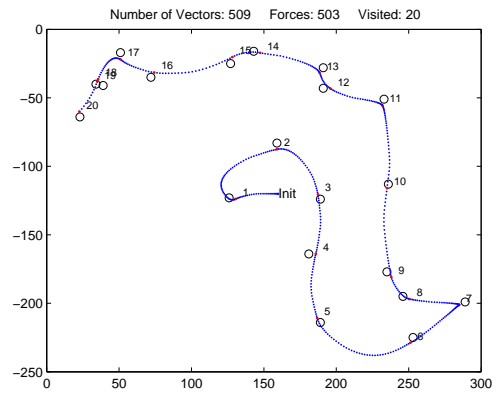


Figure 2: Map201

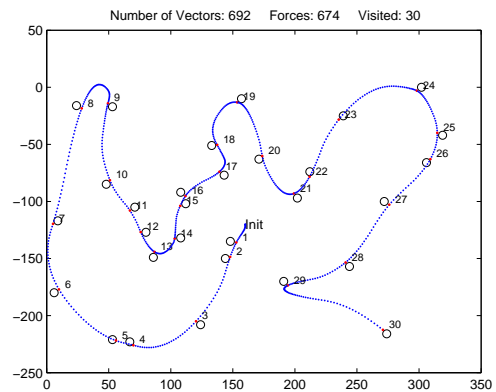


Figure 3: Map30G5