A Comparison of Constraint Handling Methods used with Particle Swarm Optimization (PSO)

Lan Guoxing

Friday, August 15, 2014

# 1 Constrained Nonlinear Optimization Problems

A general constrained nonlinear optimization problem can be stated as follows:



# 2 Particle Swarm Optimization

A PSO algorithm adopted for our study is as follows:



*  means the current generation step
*  is the velocity of individual 
*  is the position of individual 
*  are random numbers drawn from a uniform distribution between 0.0 and 1.0
*  is the inertia weight
*  is the social parameter 1, usually set to 2.0
*  is the social parameter 2, usually set to 2.0

*  is the i-th individual’s best state (position) found so far
*  is the global best state (position) found so far
*  is a constraint factor, usually set to 1.0

# 3 Constraint Handling Methods used with Particle Swarm Optimization

## 3.1 Penalty Function Method

A non-stationary, multi-stage penalty function method (PFM) implemented by Parsopoulos and Vrahatis is will be discussed in this section.

The penalty function is given below:



Where  is the original function to be optimized,  is a penalty value which is modified according to the algorithm’s current iteration number, and  is a penalty factor defined as:



Where  is a relative violated function of the constraints;  is an assignment function, and  is the power of the penalty function. The following values were used for the penalty function in all cases of the experiment.

* If , ; Else 
* If , ; Else if , ; Else if , ; Otherwise 
* The penalty value  was set to  or  depending on the complexity of the problem.

## 3.2 Superiority of Feasible Solutions (SF)

In SF, when two solutions  and  are compared,  is regarded as superior to  under the following conditions.

1.  is feasible and  is not.
2.  and  are both feasible and  has a smaller objective value (in a minimization problem) than .
3.  and  are both infeasible, but  has a smaller overall constraint violation  as computed by using (3).

## 3.3 Behavioral Memory (BehaMem)

The initial steps of the method are devoted to sampling the feasible region; only in the final step the object function  is optimized.

* Start with a random population of individuals (i.e., these individuals are feasible or unfeasible),
* Set  ( is constraint counter),
* Evolve this population to minimize the violation of the  constraint, until a given percentage of the population (so-called flip the threshold ) is feasible for this constraint. In this case

.

* Set ,
* The current population is the starting point for the next phase of the evolution, minimizing the violation of the  constraint:



During this phase, points that do not satisfy at least one of the  constraints eliminated from the population. The halting criterion is again the satisfaction of the  constraint by the flip threshold percentage of  the population.

* If , repeat the last two stepson, otherwise () optimize the objective function  rejecting unfeasible individual.

# 4 Test Problems and Results

To evaluate the performance of the methods we introduced above, several representative problems are chosen.

For each method, we test 20 times. And for each test, we let the program written in C++ run 30 times.

## 4.1 Problems with equality constraints

T1：



Results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Problem 1 | | | | | |
| Methods | Best Result | Mean Result | Worst Result | Optimal | Time（mS） |
| PFM | 1.393465 | 1.393465 | 1.393465 | 1.393465 | 28737 |
| SF | 1.396461 | 1.589053333 | 2.25972 | 1.393465 | 69248 |
| BehaMem | 1.168656 | 1.04130925 | 1 | 1.393465 | 15077 |

## 4.2 Problems with inequality constraints only

T2:



Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Problem 2 | | | | | |
| Methods | Best Result | Mean Result | Worst Result | Optimal | Time（mS） |
| PFM | -7950.926425 | -7950.956549 | -7950.961894 | -6961.81381 | 28737 |
| SF | -7973 | -7973 | -7973 | -6961.81381 | 101643 |
| BehaMem | -7973 | -7973 | -7973 | -6961.81381 | 5428 |

## 4.3 Problems with different feasible search space

T3：



Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Problem 3 | | | | | |
| Methods | Best Result | Mean Result | Worst Result | Optimal | Time（mS） |
| PFM | 1 | 1 | 1 | 1 | 13479 |
| SF | 0.700858 | 0.643776 | 0.601916 | 1 | 64053 |
| BehaMem | 0.07124 | 0.0128739 | 0 | 1 | 39312 |

T4：



Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Problem 4 | | | | | |
| Methods | Best Result | Mean Result | Worst Result | Optimal | Time（mS） |
| PFM | 1 | 1.00000165 | 1.000033 | 1 | 45536 |
| SF | 1.452811 | 2.31448025 | 6.886412 | 1 | 72149 |
| BehaMem | 0.199562 | 0.0381815 | 0.003446 | 1 | 39312 |

# 5 Conclusion

According to the results above, PFM seems to perform better than the others, if the parameters are chosen properly. However, since time is limited and I had to write all the C++ program by myself, I have only tested three methods. For every specific problem, it’s not always easy to find the proper parameters of each methods. I believe there exits other well-behaved methods according the papers I have read.

I don’t think this is an end to my research of constraint handling methods, because there are many other types of constrained problems and handling methods worthwhile to be discussed. I will go on finding a better constraint handling method in my future work.