Manual on Standard PSO Implementation

Author: Elliot Eckholm
Contributor: Professor Soumya Mohanty

Introduction

Particle Swarm Optimization (PSO) algorithms, have garnered a lot of attention in recent decades for their vast amount of applicability, flexibility and simplicity. PSO is a stochastic algorithm that is based on the behavioral properties of swarms. Since the first developed PSO there has been a lot of focus on designing the best performing PSO and thus many variants have been produced over the years. In this manual, we will be focusing on the Standard PSO variation and how to implement it, please note that everything will be written in C. This manual is organized as follows: I will briefly explain how the general PSO Algorithm works and a pseudocode outline of the sPSO Algorithm itself, and finally, how to implement a and use a modified version of the sPSO code in one's own project.

You can find the original, unmodified stand alone sPSO code here, please note I am not a contributor to this code. I have taken this code, and made some changes to it so that it is easier to implement. The modified code can be found here.

It should also be noted that there is a much more comprehensive paper on how sPSO works located here.

How does the general PSO Algorithm work?

The general PSO algorithm works by having virtual particles that make up a swarm, fly around a predefined search space and "look" for the most optimal location. If you think of each particle as a bee, then the most optimal location would be the location where there is the highest concentration of flowers for example, and the field of flowers in which the bees fly in would be called the search space. Each bee constantly checks their current location and calculates how "good" that location is. "Good" in this case means the higher the concentration of flowers, the better the location. So each bee keeps track of it's personal best location it has found so far on it's own, and is able to constantly compare it's personal best location with it's current location to see if it is better. The bee also compares it's personal best with the swarm's best location. The swarm's best location is simply the best of all the personal best locations found by any bee. If a bee is flying around and happens to find a new personal best location, and then compares this new personal best location to the swarm's best location, then the

swarm's best location is replaced by this bee's new personal best. So at any given time, one bee's personal best is also the swarm's best. The bee's keep flying around searching for a better location until the swarm's best has reach a certain criterion, such as having at least thirty flowers in a two inch by two inch area for example. PSO can be generalized to any number of dimensions, not just the four dimensions that bees are limited to, and can be applied to any problem that requires finding an optimal solution simply by changing the definition of what a "better location" means. This calculation of "better" is done by a fitness function.

Standard PSO (sPSO) Pseudo-Code:

1. Define Search Space (i.e set min and max boundaries in each dimension)
2. Initialize swarm with random initial positions and velocities inside the search space
3. FOR each Iteration:
    a. FOR each Particle: Randomize the order of updating the Particles for each iteration
        i. IF Particle is outside search space: set Particle's velocity = -0.5 Particle's velocity or just don't evaluate fitness
        ii. ELSE: Evaluate Fitness
        iii. Compare current Particle's fitness with 2 other randomly selected Particles' Fitnesses
        iv. Find particle with best fitness in this group of 3 particles
        v. IF Fitness of particle with best fitness in the group < Fitness of lBest:
            1. lBest = This Particle's position
        ii. IF Fitness of Particle's current position < Fitness of Personal Best:
            1. Personal Best = Particle's current position
            2. IF Fitness lBest < Fitness of gBest:
                a. gBest = lBest
        iii. Update Particle's velocity based on the gravitational center
        iv. Update Particle's position based on the gravitational center

How to Implement sPSO into Your own code:

The overall goal here is to be able to call the sPSO function from another function inside your main.c file and run it with a fitness function that you already have. In this GitHub repo, you will find two folders. One is called *Modified_Unitegrated_sPSO* and the other is called *sPSO_Integrated_With_MohantyCode*. We will be using the latter folder, which contains a

working example project that has it's own fitness function, and calls the sPSO function successfully in the format we want as sort of reference for the end goal here.

1. Make sure you download and install GSL first

2. *Modified_Unitegrated_sPSO* folder contains all the files you will need to call sPSO from your own project. So firstly, copy all of these files into your projects directory.

3. Inside the source file that you will call sPSO, import these files:

```
#include <gsl/gsl_vector.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_rng.h>

#include "sPSO.h"
#include "wrapper.h"
```

4. It is assumed that your project already has it's own fitness function. Please make sure that it is in the following format (you can name the function whatever you want):

```
double fitnessFunction(gsl_vector *, void *);
```

5. You should also have predefined structs that will store the PSO parameters, PSO results, and the fitness function parameters, if you have them. They should be called respectively and at the minimum, contain the following fields :

```
struct psoParamStruct {
        size_t popsize;      // Number of particles
        size_t maxSteps;     // Number of iterations
}



struct returnData {
        size_t totalIterations;  // total number of iterations
        size_t totalFuncEvals; // total number of fitness evaluations
        gsl_vector *bestLocation; // Final global best location
        double bestFitVal;  // Best fitness values found
};



struct fitFuncParams {
        size_t nDim;    // Dimensionality of the fitness function
        gsl_vector *rmin;     /*!< Minimum value of each coordinate
        gsl_vector *rangeVec; // Range of each coordinate
```

```
        gsl_vector *realCoord;  /* The unstandardized value of each coordinate is
        returned in this vector.*/
        unsigned char fitEvalFlag; // Set to 0 if fitness is infinity, else to 1

        /* Pointer to a struct that carries additional parameters that are
        specific to a fitness function. The header file of a fitness function can
        define this struct.*/
        void *splParams;
};
```

6. Now you are ready to load the structs and call sPSO. You will need to add the following code inside the function where you want to setup and call sPSO:

```
/* Define search space. NOTE: the nDim, rmin and rmax were arbitrarily set, you should
change these to correspond to your own fitness function */
        unsigned int nDim = 3, lpc;
        double rmin[nDim];
        double rmax[nDim];
        double rangeVec[nDim];

        for (int d = 0; d < nDim; d++){
                rmin[d] = -5;
                rmax[d] = 5;

        }

 // Allocate fitness function parameter struct.
        struct fitFuncParams *inParams = ffparam_alloc(nDim);


// Load fitness function parameter struct
        for (lpc = 0; lpc < nDim; lpc++){
                rangeVec[lpc]=rmax[lpc]-rmin[lpc];
                gsl_vector_set(inParams->rmin,lpc,rmin[lpc]);
                gsl_vector_set(inParams->rangeVec,lpc,rangeVec[lpc]);
        }

/* Set up pointer to fitness function. Use the prototype
declaration given in the header file for the fitness function. */
//replace fitnessFunction with whatever your fitness function is named
double (*fitfunc)(gsl_vector *, void *) = fitnessFunction;

/* Set up special parameters, if any, needed by the fitness function used.
   These should be provided in a structure that should be defined in
   the fitness function's header file.
 */
```

```
struct ptapsotestfunc_params splParams;
splParams.dummyParam = 0;

/* Pass on the special parameters through the generic fitness function parameter
struct */
inParams->splParams = &splParams;

// Set up storage for output from ptapso
struct returnData *psoResults = returnData_alloc(nDim);

// Set up the pso parameter structure
struct psoParamStruct psoParams;
psoParams.popsize=40; //number of particles
psoParams.maxSteps= 300; //number of Iterations each run

int nRuns = 2; //run sPSO 2 times

for(lpc = 0; lpc < nRuns; lpc++){

        /*wrapper_sPSO function is where sPSO is actually set up and called. This
function translates input structures you pass into it to the input structures that sPSO
requires and converts the sPSO output structures back into a format your code can use.
This wrapper_sPSO function treats sPSO as a sort of “black box” that just modifies the
inputs and outputs.*/

        wrapper_sPSO(nDim, fitfunc, inParams, &psoParams, psoResults);
}

// Print relevant information
        printf("\n\n\nTotal number of iterations %zu\n", psoResults->totalIterations);
        printf("Total number of function evaluations %zu\n",
psoResults->totalFuncEvals);
        printf("Best Location found: \n");
        for (lpc = 0; lpc < nDim; lpc++){
                printf("%f, ",gsl_vector_get(psoResults->bestLocation,lpc));
        }
        printf("\n");
        printf("Best Fitness Value: %0.35f\n", psoResults->bestFitVal);


        /*Call fitness funciton one last time to convert best location standard coords into
real coords*/
        printf("\nReal Coords:\n");
        fitfunc(psoResults->bestLocation, inParams);
        for (lpc = 0; lpc < nDim; lpc++){
                printf("%f, ",gsl_vector_get(inParams->realCoord,lpc));
        }
```

```
// Free allocated memory
ffparam_free(inParams);
returnData_free(psoResults);
```

7. To compile your code, run the following commands in your terminal:

```
gcc -c SPSO.c -I/usr/local/include
gcc -c tools.c -I/usr/local/include
gcc -c mersenne.c -I/usr/local/include
gcc -c KISS.c -I/usr/local/include
gcc -c alea.c -I/usr/local/include
gcc -c wrapper.c -I/usr/local/include
gcc -c ptapso.c -I/usr/local/include
gcc -c maxphaseutils.c -I/usr/local/include
gcc -c ptapsotestfunc.c -I/usr/local/include
gcc -c test_ptapso.c -I/usr/local/include
gcc SPSO.o tools.o mersenne.o KISS.o alea.o wrapper.o ptapso.o
maxphaseutils.o ptapsotestfunc.o test_ptapso.o -L/usr/local/lib -lgsl -lgslcblas -lm
-o test_ptapso.exe
./test_ptapso.exe
```

8. You will most likely have a fair amount of debugging to do. This is where you can use the *sPSO_Integrated_With_MohantyCode* project, which already works correctly, to compare your code to and eventually get it running. It is important to treat sPSO and the files you have copied from *Modified_Unitegrated_sPSO* as a sort of “blackbox”. All of the changes you need to make should take place in your own code.

List of Modifications from the Original sPSO Code:

- Removed all definitions of predefined fitness functions
- Deleted unnecessary source files that contained functions specific to certain fitness functions
- Switched to the Native C random number generator, the KISS rng was giving me odd errors
- Changed input of sPSO function to use a pointer function
- Added several new inputs into sPSO so that the PSO param struct, PSO results struct, and fitness function parameter struct information could be accessed throughout the code
- Instead of having switch statements for each fitness function and their corresponding parameters, I made pointer functions and structs that stored the information. This added a new level of abstraction so that any kind of fitness function could be called as long as it was in the correct format
- Had to allocate gsl_vectors and convert gsl_vector into doubles throughout various functions in the sPSO code

- Placed all sPSO setup code into the wrapper_sPSO function and called sPSO inside here so that sPSO could be called easily by simply calling the wrapper_sPSO function from within Mohanty code
- Had to convert all inputs from Mohanty code into the format sPSO required
- Had to convert all outputs from sPSO back to the format Mohanty code required

Notes:

- For questions/comments/suggestions: ellioteckholm@gmail.com

- The fitness function that the *sPSO_Integrated_With_MohantyCode* project uses is the general Rastrigin benchmark function. The global best location in real coordinates should be close to the origin: 0^{Dim}.

- *sPSO_Integrated_With_MohantyCode* was written by Professor Soumya Mohanty. I simply added the *Modified_Unitegrated_sPSO* files, which I modified myself, to his project.