Particle Swarm Optimization

Contents

1. Student Information

1.1. Personal Details

- **Name**: Adeel Ahmad
- **Email**: adeel.ahmad.3a@gmail.com
- **Homepage**: https://adl1995.github.io
- **GitHub**: https://github.com/adl1995
- **IRC (#mlpack)**: adl1995
- **Interests and hobbies**: Running, Cycling, Reading, Cooking, Automating repetitive tasks
- **Timezone**: UTC +05:00

1.2. Academic Details

- **University**: National University of Computer and Emerging Sciences, Islamabad
- **Enrollment date**: August 01, 2014
- **Graduation date (expected)**: June 15, 2018

1.3. Experience and Background

1.3.1. Programming Language Proficiency

Language	Proficiency	Experience (years)
Python	4	3
C/C++	3	2
PHP	4	3
Assembly (x86)	2	< 1
JavaScript	3	2
LaTeX	1	< 1

1.3.2. Google Summer of Code 2017

I was a participant in Google Summer of Code 2017 [1] with the OpenAstronomy organization. My project involved the design and development of a Python client for Hierarchical Progressive

Surveys (HiPS) [2], which is a scheme for describing astronomical images, source catalogues, and three-dimensional data cubes to manage large volumes of heterogeneous data. The package [3] enabled users to view astronomical figures in an interactive environment. I also maintained an extensive blog [4] throughout this period.

During the tenure of three months, I acquired a great deal of knowledge about software design principles and testing software from a user perspective. One major part of the development process was to write automated test cases and contiguously integrating code using Travis CI. Furthermore, I also composed API and user-level documentation [31] using Sphinx.

1.3.3. Full-Stack Web Developer

I have been a Full-Stack web developer for almost three years and have worked on multitude of projects ranging for SaaS to e-commerce websites. I work remotely on Upwork [5] and Fiverr [6]. My clients are either IT organizations or independent contractors. Besides web development, I also work on web automation and data scraping using `Selenium` and `BeautifulSoup`.

The skills that I acquire are through self learning and consistency. My GitHub profile [7] contains some of my other open source work.

1.3.4. Open Source Contributions

Apart from the hips repository [3], where I have been contributor till GSoC 2017, I have made small contributions to these projects: OpenCV [8], libLAS [9], Jailhouse [10], Jupyter notebook [11], and axios [12].

1.3.5. Relevant Coursework

My interest in machine learning sparked from taking my college courses: Data Mining and Deep Learning, which contained similar content as Stanford's CS229 [32] and CS231n [33], respectively. The course content included but was not limited to: gradient descent (vanilla and mini-batch), linear, softmax, and SVM classifiers, Bernoulli distribution, convolutional neural networks, and recurrent neural networks (RNNs, LSTM). I also made a blog post explaining the various activation functions used in neural networks [13].

Apart from this, I am also enthusiastic about computer vision. My final year project involves implementing a structure from motion pipeline using a monocular camera [14]. To learn all the intricate steps involved in this process, I followed the Robotics: Perception course on Coursera [15]. Image processing algorithms have always fascinated me, to quench this fascination I implemented numerous filters (Gaussian, Sobel, Prewitt, Laplacian), edge detectors (Canny, Marr Hildreth), morphological operators (Convex hulling, Erosion, Dilation), and object detectors (Generalized Hough transform, simple convolution) [16, 17]. These were implemented from

scratch making use of the `NumPy` Python library. The most interesting algorithm I implemented was Generalized Hough Transform [18]. I was amazed to see how something simple as the equation of line could lead to detection of objects in images.

My résumé [19] contains a more detailed overview on my interests and courses.

1.3.6. Other Questions

> *What are your long-term plans, if you have figured those out yet? Where do you hope to see yourself in 10 years?*

There is a major technical revolution coming in the near future. I believe this breakthrough will come from the domain of artificial intelligence.

However, instead of just speculating the future, I want to play a part in shaping it. So, I see myself as someone who has a good grasp in the machine learning domain, and is constantly applying his skills to contribute towards its development.

> *Describe the most interesting application of machine learning you can think of, and then describe how you might implement it.*

Image synthesis is the mosting interesting application of machine learning I can think of. It applies deep learning techniques to generate an image from a piece of text. This would require training two neural networks, a generative network, which tries to sketch the shape and colors of the object based on the given text description, and a discriminator network that tries to judge whether the generated image was real or fake. Thus, both neural networks will work towards perfecting their tasks.

> *Both algorithm implementation and API design are important parts of mlpack. Which is more difficult? Which is more important? Why?*

In my opinion, API design is the more difficult and important part of mlpack, or any library, for that matter. Details regarding algorithm implementation, in most cases, can be found in research papers.

Considering C++, there are numerous API design patterns that can be followed to empower user with the ability to tweak the underlying details. For example, a policy-based design can be adopted in case of PSO to select its variant on the basis of template parameter.

2. Project Proposal

Particle swarm optimization is an optimization technique inspired by the social flocking of birds. The problem consists of a population of particles, where each particle is assigned a unique position and velocity, which the algorithm tries to update in a way that allows it to reach the global best solution.

2.1. Objective

There exist several variants to particle swarm optimization. In the algorithm proposed by Eberhart and Kennedy [20], the group of particles are represented as a swarm, where all particles are attracted towards the global optima. Each particle keeps track of its best position, and moves through the solution space, evaluating its position along each step. Secondly, the particle keeps track of the global best position that was found by one member of the swarm, and this information is available to all swarm particles.

A solution is already proposed for the global best PSO variant for unconstrained problems [21]. So, one objective of this project is to extend this implementation to handle constrained problems, for which an API is proposed below. Another objective is to implement two other variants of PSO. I propose to implement the local best PSO and PSO with Velocity Adaption, proposed by S. Helwig et al. [22].

The implementation has to be backed up with extensive test cases, including a diverse set of problems from convex optimization machine learning algorithms e.g. logistic regression and neural networks to low dimensional optimization functions. An extensive set of optimization problems is currently provided by mlpack [30]. These can be used in addition to the previously proposed problems to benchmark our implementation. For testing constrained optimization, new problems would have to be introduced, or an infrastructure would have to be devised which would allow adding constraints to unconstrained problems.

If the implementation is finished before the proposed timeline, support for initialization techniques will be added, as the convergence of PSO greatly depends on the parameter's tuning. One solution is to try and find the best set of hyperparameters using evolutionary algorithms. Other solutions have been proposed in [23].

2.2. Problem Description

Particle swarm optimization is similar to genetic algorithms, except it does not provide genetic operators like crossover and mutation. Instead, it updates particles with the internal velocity (equation provided in section 3). In the initially proposed implementation of the algorithm, particle velocity was updated on the basis of its nearest-neighbor, however, it was found that this technique made the particles settle on a unanimous, unchanging direction. To account for this undesired behavior, a stochastic variable called “crazniess” was introduced. In the implementation proposed by Eberhart and Kennedy [20], both the particle position and velocity are initialized with uniform random numbers.

The current implementation treats the group of particles as a swarm, which can be visually compared with a flock of birds. Each particle keeps track of its best position over the entire run, this is referred to as its `pbest` value. This conceptually resembles autobiographical memory, as each individual remembers its own experience. Similarly, velocity adjustment associated with the best position `pbest` has been called “simple nostalgia” in that an individual tends to return to the place that most satisfied it in the past. The global best position or `gbest` is conceptually similar to publicized knowledge, or a group norm, which individuals seek to attain.

The cognitive (`pbest`) and social (`gbest`) elements have an associated acceleration, which are set following a heuristic. These variables produce a variation in velocity at each iteration and try to converge on a point that is globally optimum.

2.3. Handling Constrained Problems

Similar to other nature-inspired techniques like evolutionary computation, PSO in its original form is designed to handle unconstrained problems [24] i.e. they lack the ability to include feasibility information of solutions in their fitness values. The common approach to deal with constraints is by using a penalty function, which aims to decrease the fitness of infeasible solutions. Another approach is to create a distinction between objective functions and constraints, one of this approach was proposed by Toscano and Coello [25] for the global best PSO where the leader is chosen based on the lowest number of violated constraints.

It is known that for constrained problems the probability of a particle leaving the bounded-search space largely depends on its velocity. Thus, if we are somehow able to control / decrease it, we can enforce the particles to remain within the boundaries enforced by the search space. However, this can lead to slow progress of the algorithm. The velocity adaption virant of PSO proposed in [26] shows that a large progress can be achieved with higher velocities. The low velocity is only attained if the algorithm observes to have difficulties finding better solutions. Adaption of velocity is similar to the step size in evolution strategies. A success factor is introduced which accounts for the amount of progress made during the last iterations. It is considered a success when the particle’s private guide (best position) is set to its current position. If the new particle position and the private guide are of equal fitness, the private guide

is updated with a probability of $1/2$. If an improvement is found in the last iterations, the velocity is increased, otherwise, it is decreased.

The success rate of the algorithm for nnn iterations is given by $SuccessCounter/(n*m)$, where m is the number of particles. If the observed success rate is higher than a given threshold (called its `SuccessProbability`) the velocities are increased by a factor of 2, otherwise, the velocities are scaled down by a factor of $1/2$.

2.3.1. Implementation

There a two main design choices in the implementation of constrained or bounded problems. One solution is to make use of lambda functions. This provides user the flexibility to define constraints and pass them to the optimizer. A lambda function can be defined as below:

```
auto constraint = [](double x) { return x < 3; };
```

The user can either define multiple constraints by providing multiple lambda functions, which the optimizer can handle using variadic templates [27], by making use of type expansion. An example is shown below:

```
template<typename T>
bool constraintCheck(T v) {
  return v(3);
}

template<typename T, typename... Args>
bool constraintCheck(T constraint, Args... args) {
  return constraint(3) && constraintCheck(args...);
}

int main() {
  auto constraint1=[](int x){ return x < 3; };
  auto constraint2=[](int x){ return x > 3; };

  constraintCheck(constraint1, constraint2);
}
```

Another option for the user is to provide multiple constraints in a single lambda function, such as:

```
auto constraint = [](double x, double y) { return x < 3 && y > 2; };
```

However, this might restrict the user in that they won't be able to modify constraints once the algorithm has begun execution.

For checking the presence of constraints in a constrained problem, a `HasConstraints` function could be provided that returns true if constraints are present. The same could be achieved by using substitution failure is not an error (SFINAE) [34] and `std::enable_if` to detect if a class implements a specific method and call it accordingly.

2.4. API Design

mlpack employs a policy-based design among numerous optimizers e.g. CMAES [28] and Frank-Wolfe [29]. This meta-programming technique allows the algorithm to choose the variant on basis of its template parameter. Additionally, each variant requires a different initialization technique. For example, the constriction factor variant of PSO requires the cognitive and social acceleration for computing k i.e. the constriction factor. Whereas, variants such as inertia weight do not require such initialization. It only introduces a constant ω to control the impact of previous history of velocities on the current velocity.

Since the constriction based PSO requires computing the constriction factor before updating the velocity, our currently proposed solution in [21] is to provide an `Initialize` class method, which computes the required value only once, thus saving computational cost. In case of inertia weight variant of PSO, this method is provided with dummy parameters.

3. Current Progress

I started working on the implementation of global best PSO for unconstrained problems. The implementation currently provides two variants, (1) PSO with inertia weight, and (2) PSO with constriction factor.

The first variant makes use of a parameter called omega (ω), which is used during velocity update. The equation is given by:

$$v_i(t+1) = \omega * v_i(t) + c1 * rand() * (x_{pbest_i} - x_i) + c2 * rand() * (x_{gBest_i} - x_i)$$

The second variant introduces a parameter called the constriction factor (k), this can be computed as follows:

$$k = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}$$

where $\phi = c1 + c2,\ \phi > 4$.

$c1$ and $c2$ represent the cognitive and social acceleration, respectively.

4. Timeline

Time Period	Milestones
April 24- May 13 **(Community Bonding Period)**	● Get an in-depth understanding of mlpack's codebase. ● Familiarise myself with C++ design patterns, such as SFINAE and lazy / eager evaluation ● Read research journals, articles, blogs to understand the extent to which PSO can be applied to solve problems
Phase 1 begins	
May 14 - May 21 (1 week)	● Implement the lbest PSO variant for unconstrained problems with test cases
May 22 - May 29 (1 week)	**Final exams**
May 30 - June 05 (1 week)	● Implement PSO with Velocity Adaption with extensive documentation and test cases
June 06 - June 14 (~1 week)	● Create an API which can handle constrained problems for the implemented PSO variants
Phase 1 ends **Phase 2 begins**	

June 15 - June 22 (1 week)	● Create problems of varying dimensionality with constraint bounds and test the constraint based API
June 23 - June 30 (1 week)	● Implement an initialization technique that produces good results for a variety of problems
July 01 - July 13 (~2 weeks)	● Provide further test case for constrained problems ● Understand and implement other variants of PSO e.g. Binary PSO and multi-objective PSO
Phase 2 ends	
July 14 - July 21 (1 week)	● Optimize the implementation, especially steps involving velocity and position update
July 22 - July 31 (~1 week)	● Benchmark the performance on the basis of different factors e.g. accuracy, execution time, initialization techniques, etc.
Aug 01 - Aug 07 (1 week)	● Provide diverse examples to allow users get quickly started
Aug 08 - Aug 14 (~1 week)	● Write API and user-level documentation

References

[1] HiPS client for Python
https://summerofcode.withgoogle.com/archive/2017/projects/5440053895495680
[2] Hierarchical progressive surveys
https://www.aanda.org/articles/aa/pdf/2015/06/aa26075-15.pdf
[3] Python astronomy package for HiPS, https://github.com/hipspy/hips
[4] Blog for GSoC 2017, https://adl1995.github.io/category/gsoc.html
[5] Upwork profile, https://www.upwork.com/freelancers/~018e56b8591046f889
[6] Fiverr profile, https://www.fiverr.com/adl1995
[7] GitHub profile, https://github.com/adl1995
[8] OpenCV commits for adl1995,
https://github.com/opencv/opencv/commits/master?author=adl1995
[9] libLAS commits for adl1995,
https://github.com/libLAS/libLAS/commits/master?author=adl1995
[10] Jailhouse commits for adl1995,
https://github.com/siemens/jailhouse/commits/next?author=adl1995
[11] Jupyter notebook commits for adl1995,
https://github.com/jupyter/notebook/commits/master?author=adl1995
[12] Axios issue “Getting ‘Cross-Origin Request Blocked’ on a GET request”,
https://github.com/axios/axios/issues/853
[13] An overview of activation functions used in neural networks,
https://adl1995.github.io/an-overview-of-activation-functions-used-in-neural-networks.html
[14] Structure from Motion pipeline, https://github.com/reconstruct3d/SfM
[15] Robotics: Perception | Coursera, https://www.coursera.org/learn/robotics-perception
[16] An implementation of Marr Hildreth and Canny edge detector,
https://github.com/adl1995/edge-detectors
[17] Multiple Choice Questions detector, https://github.com/adl1995/exam-mcq-recognition
[18] An implementation of Generalised Hough transform,
https://github.com/adl1995/generalised-hough-transform
[19] Résumé, https://adl1995.github.io/personal/resume.pdf
[20] Particle swarm optimization, by Kennedy et al.,
http://ieeexplore.ieee.org/document/488968/?reload=true

[21] Introduce Particle Swarm Optimization for unconstrained problems, https://github.com/mlpack/mlpack/pull/1225
[22] Particle Swarm Optimization with Velocity Adaptation, by S. Helwig et al., http://ieeexplore.ieee.org/document/5329547
[23] Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method by Parsopoulos et al., https://www.math.upatras.gr/~vrahatis/papers/volumes/ParsopoulosV02b_Advances_in_Intelligent_Systems_FS_EC_A_Grmela_et_al_eds_pp216-221_WSEAS_2002.pdf
[24] Looking Inside Particle Swarm Optimization in Constrained Search Spaces, by Jorge et al. https://pdfs.semanticscholar.org/5962/a6dba4776870f6bf2ad79b77e67f4dc1f628.pdf
[25] A Constraint-Handling Mechanism for Particle Swarm Optimization, by Gregorio et al. https://pdfs.semanticscholar.org/be9a/76f0d07c4dc137a7e2f1b9d0f857b56eac89.pdf
[26] Velocity Adaptation in Particle Swarm Optimization by Helwig et al., http://ieeexplore.ieee.org/document/5329547
[27] Variadic template, https://en.wikipedia.org/wiki/Variadic_template
[28] CMAES, https://github.com/mlpack/mlpack/tree/master/src/mlpack/core/optimizers/cmaes
[29] Frank-Wolfe, https://github.com/mlpack/mlpack/tree/master/src/mlpack/core/optimizers/fw
[30] Optimization problems provided my mlpack https://github.com/mlpack/mlpack/tree/master/src/mlpack/core/optimizers/problems
[31] Documentation for hips package, https://hips.readthedocs.io/en/latest/api.html
[32] CS229: Machine Learning, http://cs229.stanford.edu
[33] CS231n: Convolutional Neural Networks for Visual Recognition, http://cs231n.stanford.edu
[34] Substitution failure is not an error, https://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error