Artificial Intelligence Project

Section 4: MOPSO Algorithm

Abolfazl Aghdaee 9811152202

Introduction:

Multi-objective optimization problems Optimizer (MOPSO) are very common in engineering and other areas of research, such as economics, finance, production scheduling, and aerospace engineering. It is very difficult to solve these problems because they usually involve several conflicting objectives. Generally, the optimal solution of MOPSO is a set of optimal solutions (known as a Pareto optimal set), which differs from the solution of single-objective optimization (with only one optimal solution). Some classical optimization methods (weighted methods, goal programming methods, etc.) require the problem functions to be differentiable and are required to run multiple times with the hope of finding different solutions. In recent years, the multi-objective optimization evolutionary algorithm (MOEA) has become a popular method for solving MOPSO, and it has garnered scholarly interest around the world. Many representative MOEAS, such as multiple objective genetic algorithms (MOGA), non-dominated sorting genetic algorithm II (NSGA-II), strength Pareto evolutionary algorithm (SPEA), and Pareto archived evolution strategy (PAES), have been presented.[1]

MOPSO (simply PSO), is one of the most important intelligent optimization algorithms within the field of Swarm Intelligence. Introduced by James Kennedy and Russell C. Eberhart in 1995, this algorithm draws inspiration from the social behavior of animals such as fish and birds, which live together in small and large groups. In the PSO algorithm, members of the population directly interact with each other and, through the exchange of information and recollection of past successful experiences, work towards solving the problem.

[1]ttps://www.researchgate.net/publication/3949342_MOPSO_A_proposal_for_multiple_objective_particle_swarm_optimization

Given the remarkable success of the PSO algorithm in solving single-objective optimization problems, many scientists and researchers have attempted to utilize this algorithm for solving multi-objective problems. Multiple versions of the PSO algorithm have been proposed for multi-objective optimization. One of the most well-known algorithms introduced in this context is the work by Professor Coello Coello and his colleagues. They named their algorithm MOPSO, a term commonly used exclusively for this algorithm. MOPSO was introduced in 2004 in an article published in the IEEE Transactions on Evolutionary Computation journal.[2]

Unlike or contrary to the genetic algorithm, MOPSO does not have any evolutionary operators such as mutation and crossover. Each element of the population is referred to as a particle (analogous to a chromosome in GA). In fact, the PSO algorithm is composed of a specific number of particles that randomly take initial values. For each particle, two values, namely position and velocity, are defined and modeled as a position vector and a velocity vector, respectively. These particles iteratively move in n-dimensional space to explore new possible options by calculating the optimization value as a criterion.

The dimension of the problem space is equal to the number of parameters in the target function for optimization. A memory is assigned to store the best position of each particle in the past, and another memory is allocated to store the best position encountered among all particles. Based on the experience gained from these memories, particles decide how to move in the next iteration. In each iteration, all particles move in the n-dimensional space until,

[2] Transactions on Evolutionary Computation journal

eventually, the global optimal point is found. The particles update their velocities and positions based on the best absolute and local solutions.

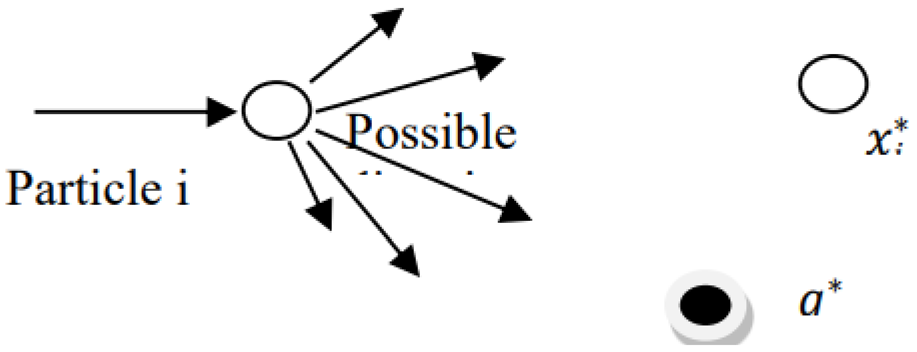


Figure1
A schematic representation of the motion of a particle, moving towards globe best g* and current best x_i^* for each particle[3]

Multi-Objective Particle Swarm Optimization Approach:

The analogy of particle swarm optimization with evolutionary algorithms makes evident the notion that using a Pareto ranking scheme could be the straightforward way to extend the approach to handle multiobjective optimization problems. The historical record of best solutions found by a particle (i.e., an individual) could be used to store nondominated solutions generated in the past (this would be similar to the notion of elitism used in evolutionary multiobjective optimization). The use of global attraction mechanisms combined with a historical archive of previously found

[3] Yang, X.S. (2010). Engineering Optimization an Introduction with Metaheuristic Applications John Wiley & Sons, 19-22.

nondominated vectors would motivate convergence toward globally nondominated solutions.
Therefore, this essay is based on the idea of having a global repository in which every particle will deposit its flight experiences after each flight cycle. Additionally, the updates to the repository are performed considering a geographically-based system defined in terms of the objective function values of each individual. This technique is inspired on the external file used with the Pareto Archive Evolution Strategy (PAES) [8]. The repository previously mentioned is used by the particles to identify a leader that will guide the search. We implemented a mechanism such that each particle may choose a different guide. Our mechanism is based on the generation of hypercubes which are produced by dividing the search space explored.

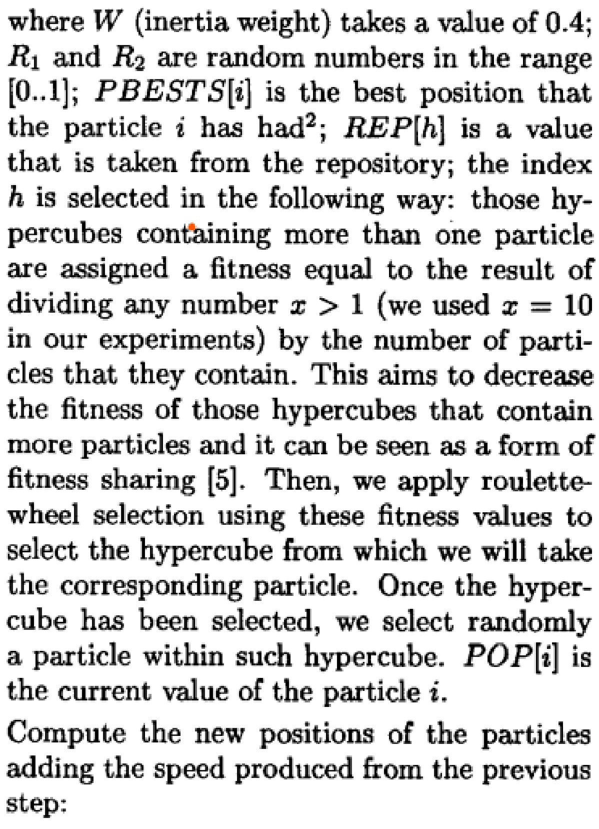
The algorithm of MOPS0 is the following:

```
1. Initialize the population POP:
    -   For i=0 to MAX    # MAX = number of particles
    -    Initialize POP[i]

2. Initialize the speed of each particle:
    -   For i = 0 to MAX
    -   VEL[i] = 0

3. Evaluate each of the particles in POP.

4. Store the positions of the particles that represent nondominated vectors in
the repository REP
```

5. Generate hypercubes of the search space explored so far, and locate the particles using these hypercubes as a coordinate system where each particle's coordinates are defined according to the values of its objective functions.

6. Initialize the memory of each particle (this memory serves as a guide to travel through the search space. This memory is also stored in the repository).
   - For i = 0 to MAX
   - PBESTS[i] = POP[i

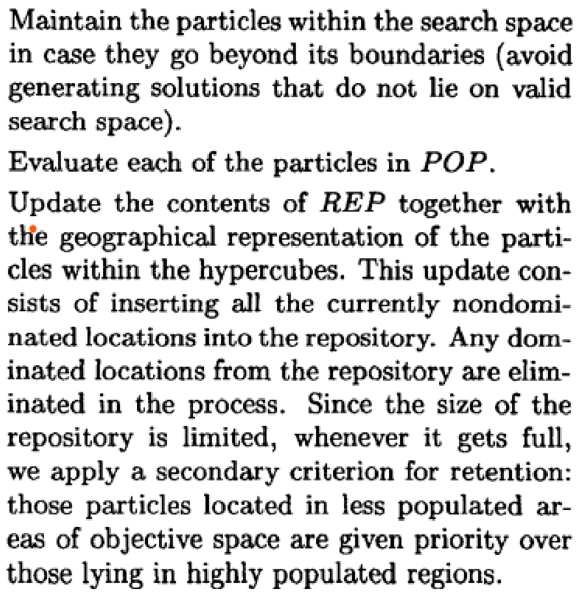
8. WHILE maximum number of cycles has not been reached DO:
   - Compute the speed of each particle1 using the following expression:
     VEL[i] = W x VEL[i] + R_1 x (PBESTS[i] - POP[i]) + R_2 x (REP[h] - POP[i])



where W (inertia weight) takes a value of 0.4; R_1 and R_2 are random numbers in the range [0..1]; $PBESTS[i]$ is the best position that the particle i has had[2]; $REP[h]$ is a value that is taken from the repository; the index h is selected in the following way: those hypercubes containing more than one particle are assigned a fitness equal to the result of dividing any number $x > 1$ (we used $x = 10$ in our experiments) by the number of particles that they contain. This aims to decrease the fitness of those hypercubes that contain more particles and it can be seen as a form of fitness sharing [5]. Then, we apply roulette-wheel selection using these fitness values to select the hypercube from which we will take the corresponding particle. Once the hypercube has been selected, we select randomly a particle within such hypercube. $POP[i]$ is the current value of the particle i.

Compute the new positions of the particles adding the speed produced from the previous step:

POP[i] = POP[i] + VEL[i]



Maintain the particles within the search space in case they go beyond its boundaries (avoid generating solutions that do not lie on valid search space).

Evaluate each of the particles in *POP*.

Update the contents of *REP* together with the geographical representation of the particles within the hypercubes. This update consists of inserting all the currently nondominated locations into the repository. Any dominated locations from the repository are eliminated in the process. Since the size of the repository is limited, whenever it gets full, we apply a secondary criterion for retention: those particles located in less populated areas of objective space are given priority over those lying in highly populated regions.

When the current position of the particle is better than the position contained in its memory, the particle’s position is updated using:

PBESTS[i] = POP[i]

The criterion to decide what position from memory should be retained is simply to apply Pareto dominance (i.e., if the current position is dominated by the position in memory, then the position in memory is kept; otherwise, the current position replaces the one in memory; if neither of them is dominated by the other, then we select one of them randomly).

8. END WHILE.

Here is a brief explanation of how this algorithm works in Persian(فارسی):

الگوریتم MOPSO (بهینه‌سازی گروه ذرات چند هدفه) یک الگوریتم بهینه‌سازی چند هدفه بر اساس مفهوم بهینه‌سازی چشمه ذرات است. هدف این الگوریتم یافتن مجموعه‌ای از راه‌حل‌هاست که تجارت میان اهداف متعارض را نمایان کنند.

روش کار:

1. جمعیتی از ذرات موجودات (هرچیزی)با موقعیت و سرعت تصادفی در دامنه جستجوها ایجاد کنید.
2. مقادیر هدف را بر اساس اهداف مسئله برای هر ذره ارزیابی کنید یا همان تابع فیتنس.
3. موقعیت بهتر شخصی یا همان personal برای هر ذره بر اساس موقعیت فعلی و مقادیر هدف به‌روزرسانی شود.
4. موقعیت بهتر گلوبال بر اساس موقعیت بهتر شخصی تمام ذرات به‌روزرسانی شود.
5. سرعت و موقعیت ذرات بر اساس موقعیت بهتر گلوبال و شخصی به‌روزرسانی شود.
6. مراحل 2 تا 5 برای تعدادی مشخص از تکرارها یا تا رسیدن به شرایط همگرایی تکرار شود.
7. در نهایت، مجموعه راه‌حل‌های غیرمسلط یافته شده در تمام تکرارها جبهه پارتو را نمایان می‌کند

تفاوت های الگوریتم ژنتیک با الگوریتم MOPSO از دیدگاه خودم

اول روش تولید پاسخ های جدید در دو الگوریتم متفاوت هست. در MOPSO با توجه به موقعیت کنونی، بهترین موقعیتی که خود ذره تا حالا داشته و موقعیت بهترین ذره در جمعیت، پاسخ جدید با توجه به یک فرمول مشخص تولید میشه! در ژنتیک دو پاسخ انتخاب میشن و دو عملگر ترکیب روی این پاسخ ها اعمال میشه. حالا در اینجا عملگر جهش روی یک پاسخ هم در ژنتیک و هم در MOPSO می تونه استفاده بشه

دومی هم روش تشکیل جبهه پارتو در دو الگوریتم متفاوت هست. به طور کلی در ژنتیک از تمام اطلاعات جمعیت که وجود داره برای تشکیل پارتو استفاده میشه! در MOPSO از بخشی از اطلاعات موجود که اون هم به تصادف یا با یک مکانیزم خاصی انتخاب میشن، جبهه پارتو تشکیل میشه.