Multi-objective Optimization with PSO

1 Problem statement

Multi-objective optimization is a class of problems with solutions that can be evaluated along two or more incomparable or conflicting *objectives*. These types of problems differ from standard optimization problems in that the end result is not a single "best solution" but rather a set of alternatives, where for each member of the set, no other solution is completely better (the *Pareto set*). Multi-objective optimization problems occur in many different real-world domains, such as architecture (stability vs. cost), and automobile design (performance vs. fuel efficiency) and as such are a very important problem domain.

2 Related work

Solving multi-objective optimization problems can be done in several ways. The simplest is to construct a single meta-objective function by taking a weighted sum of the individual objectives. Such an approach is limited, however, as it is limited to a convex subset of all non-dominated solutions. This exclusion may skip over important representative candidate solutions that would be relevant to the end user.

A better approach, adopted in the evolutionary computation literature [1] [3], is to Pareto rank candidate solutions, and keep an archive of all non-dominated such. In this way it's possible to explore the entire Pareto front without any *a priori* knowledge about the problem. Such approaches have been explored in the context of other population based approaches, such as *particle swarm optimization* (PSO), and it's the current state of the art in multi-objective optimization with PSO that I'd like to explore in this project.

3 Approach

I implemented a multi-objective particle swarm optimization algorithm using the formulation outlined in [2]. PSO is a population-based optimization approach. The basic idea behind the algorithm is to use a collection of "particles" to explore the fitness landscape of a particular problem. Each particle is a vector that describes a candidate solution, and can be evaluated (in the multi-objective case) along several quality dimensions (or, equivalently, with several fitness functions). The algorithm is iterative, and at each iteration each particle "moves" through the fitness landscape according to it's current fitness values as well as those of nearby particles, and the swarm as a whole. The premise being that by emulating flocking-type behaviors, an efficient parallel search of the fitness landscape can be performed. The exact steps of the PSO algorithm for the single-objective case are:

1. Initialize the swarm
2. For each particle in the swarm:
    (a) Select leader
    (b) Update velocity
    (c) Update position
3. Update global best
4. Repeat

When tackling multi-objective problems however, a few modifications must be made. First, the objective is to find not one "global best" solution, but a set of solutions comprising the Pareto Front. To do this, an *archive* of non-dominated solutions is kept, where all non-dominated solutions found at each iteration are stored. The MOPSO algorithm steps are:

1. Initialize the swarm & archive
2. For each particle in the swarm:
    (a) Select leader from the archive
    (b) Update velocity
    (c) Update position
3. Update the archive of non-dominated solutions
4. Repeat

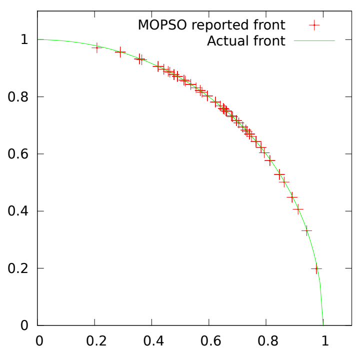

Figure 1: 100 sample Front found by MOPSO using epsilon dominance.

Since the front is usually continuous, some further criteria must be used to decide which non-dominated solutions to keep in the finite archive. Generally, the criteria enforces some diversity measures, the intuition being that a more diverse front will ensure good coverage, as opposed to clustering the reported solutions in one area. One technique for encouraging diversity is to use ϵ-dominance, where the area dominated by a given point is increased by a small constant. Since nearby points will be considered dominated under this new definition, it has the effect of spreading out those solutions kept in the archive.

For this project, I implemented a standard MOPSO using an archive with the standard definition of dominance, as well as ϵ-dominance. The results are given in the next section.

4 Evaluation

The multi-objective problem I used consisted of two fitness functions, with an artificially constructed boundary at unit distance from the origin, beyond which both fitness values were set to -1. This creates a convex circular front.

Evaluation of multi-objective optimizers is different from the evaluation of standard optimization techniques. Rather than directly comparing the quality of candidate solutions, the quality of the overall solution set should be evaluated by how well it represents the true Pareto front. One qualitative measure of this is the variance among the reported solutions. After running both normal dominance and ϵ-dominance versions, it was found that both had similar variance, with ϵ-dominance slightly larger (0.181 versus 0.177, averaged over 30 runs).

A more intuitive visualization is given in Figure 1. This shows the multi-objective function overlaid by the front found by the swarm.

5 Discussion

As can be seen in Figure 1, the MOPSO algorithm does a fairly good job of representing the front, with some difficulty at the extreme edges, where either f_1 or f_2 approach 1. This makes sense because those areas are near the edges of the random initialization, and therefore have a lower probability of being fully explored.

The fact that ϵ-dominance reports increased variance is also what we'd expect, since each solution in the archive "covers" a larger area of the fitness landscape.

One avenue for potential future work would be to analyze a larger set of test problems, as well as different mechanisms for promoting diversity. The concept of ϵ-dominance is presented as one of the best techniques in terms of an efficiency/quality trade off, but it requires some parameter tweaking to work well for a specific problem. Techniques such as fitness sharing, which scales the objective values of a point inversely with the number and nearness of its neighbors, might be able to achieve similar diversity results without any problem specific parameter tuning (at the expense of increased running time).

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[2] M. Reyes-Sierra and C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[3] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *EUROGEN 2001*. CIMNE, 2002.