IT3105: Project 3 – Particle Swarm Optimization

Due on Wednesday, Novemeber 24, 2013

Lecturer: Jo Skjermo, Keith Downing

Pablo Liste Garcia & Dominik Horb

Contents

1 Architecture Description

The idea for the architecture of the system we devised was to have generalized Particles that would be able to store solutions to very different kinds of constraint based problems and a basic *ParticleSwarm* class, that would hold similarities between the different implementations.
The hardest part for us to understand was the difference between binary particles and ones that could hold floating point solutions and how the vectors could be used to represent the solutions to the given problems. As we were just able to implement the PSO for the Knapsack problem due to time constraints, we couldn't test how the general structure we have would hold up to cater to more different problem styles. The system architecture would probably need some more tweaking, we however believe, that it goes in the right direction to be very versatile.

2 Task 3 – Simple Knapsack Problem

As you can see in Figure 1, the number of packages in a solution that is found by our implementation is usually around 30 with the highest fitness going slightly above 200 during good runs.
In Figure 2 you can see the plots of three different runs of the algorithm. For us it seems like the different values for c1 and c2 do not make a big difference in the outcome of the algorithm, although this is just a not very scientific observation based on the runs we made by hand.

3 Task 3 – Multiple Constraints Knapsack Problem

The addition of another constraint to the knapsack problem wasn't very difficult for us. As you can see in Figure 3 we just needed to add another check for the *VOLUME_LIMIT* instead of just the *WEIGHT_LIMIT*. This could probably be solved more cleanly, so that future additions of constraints could be added easier, but was sufficient for now.

```
Packages in best solution:
Value: 7.30251135791, Weight: 12.7327994777, Volume: 4.0
Value: 5.76492711048, Weight: 57.1879434508, Volume: 5.0
Value: 0.495317906434, Weight: 3.43710613351, Volume: 3.0
Value: 6.1701704794, Weight: 18.083197733, Volume: 1.0
Value: 0.726586714447, Weight: 16.5000167102, Volume: 3.0
Value: 9.99505099161, Weight: 4.27655268474, Volume: 5.0
Value: 6.2187743344, Weight: 37.1407155989, Volume: 3.0
Value: 5.88379788085, Weight: 88.5940904905, Volume: 3.0
Value: 5.25902870043, Weight: 9.09410823592, Volume: 3.0
Value: 7.98150387315, Weight: 5.08868159753, Volume: 3.0
Value: 0.207243524083, Weight: 44.998404325, Volume: 1.0
Value: 6.7242235471, Weight: 22.2791697705, Volume: 1.0
Value: 7.65820717721, Weight: 63.8961535034, Volume: 4.0
Value: 8.61850796453, Weight: 32.5107772723, Volume: 1.0
Value: 7.05058655206, Weight: 11.3266547226, Volume: 1.0
Value: 5.70452363561, Weight: 46.7219980762, Volume: 5.0
Value: 4.21869466296, Weight: 26.7061575878, Volume: 3.0
Value: 2.77575404294, Weight: 86.3041796499, Volume: 5.0
Value: 8.89622070816, Weight: 16.9665124264, Volume: 2.0
Value: 6.21239193962, Weight: 35.0962139535, Volume: 2.0
Value: 8.19762960423, Weight: 10.1952783091, Volume: 5.0
Value: 2.99264669383, Weight: 2.78554242048, Volume: 1.0
Value: 6.94776088459, Weight: 13.1307012028, Volume: 2.0
Value: 6.64014969239, Weight: 24.7036879454, Volume: 2.0
Value: 4.03723671619, Weight: 24.7865453009, Volume: 1.0
Value: 7.26628597628, Weight: 99.8682797165, Volume: 4.0
Value: 0.819744152312, Weight: 4.52527653988, Volume: 3.0
Value: 9.33875277213, Weight: 64.2389241489, Volume: 2.0
Value: 7.8193753921, Weight: 16.0858245809, Volume: 3.0
Value: 9.73182033646, Weight: 74.3929790294, Volume: 5.0
Number of packages: 30
BUILD SUCCESSFUL (total time: 26 seconds)
```

Figure 1: Output of a knapsack run with just a weight constraint (the volume output is just visible in general).

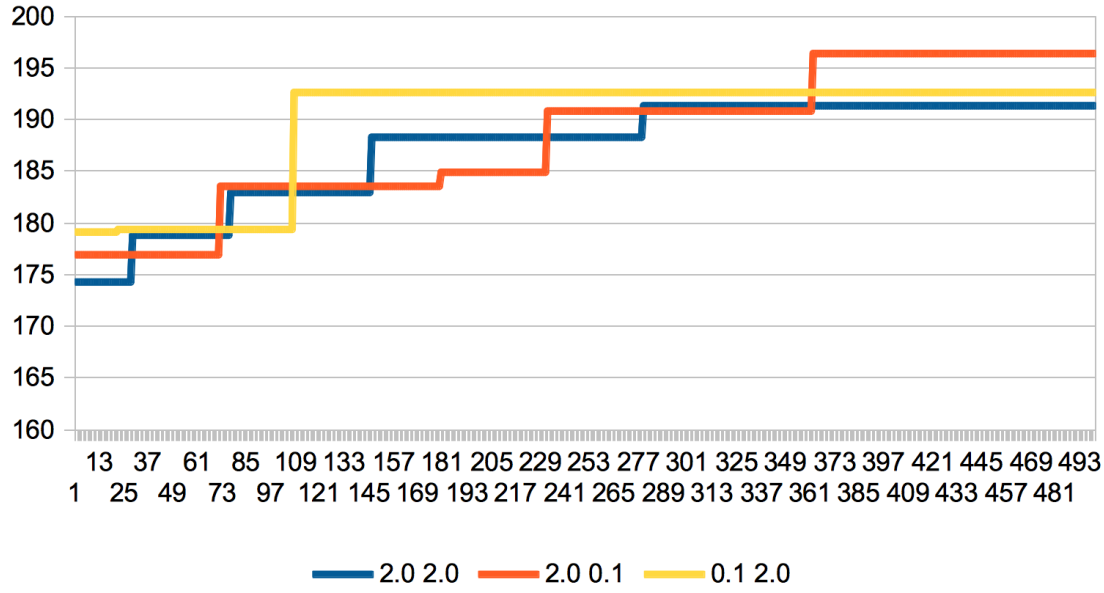

Figure 2: Plot of three different Knapsack Problem runs.

Figure 3: Adding another constraint to the knapsack problem.

```
while (weightSum < WEIGHT_LIMIT && volumeSum < VOLUME_LIMIT) {
  selectedPackage = packages.get(this.random.nextInt(packages.size()));
    double newSum = weightSum + selectedPackage.getWeight();
    double newVolumeSum = volumeSum + selectedPackage.getVolume();
    if (!selectedPackages.contains(selectedPackage)) {
      if (newSum <= WEIGHT_LIMIT && newVolumeSum <=VOLUME_LIMIT) {
        selectedPackages.add(selectedPackage);
        valueSum += selectedPackage.getValue();
      }
      weightSum = newSum;
      volumeSum = newVolumeSum;
    }
  }
}
```