CS572 Project #1a Report

Heyan Huang

February 4, 2014

1 Algorithm Descriptions

1.1 Hill Climbing

Hill Climbing is a mathematical optimization technique which belong to the family of local search. It starts from a random solution to a problem and trying to find a better solutions by incrementally change one element of the solution. Hill Climbing is simply a greedy searching algorithm. It selects the best solution from nearby tried selection space, and does this step repeatedly until eventually it finds a best optimum.

The advantage for this algorithm is that it is simple and easy, but it usually ends up with local optimum only, and misses out the global optimum because it always accepts only the best solutions.

Data Structure:

- In this project, I have used double precision float variables.
- I used a vector of integer of length 15 to record the mutated dimensions, and
- a vector of double (of length 30) to record the mutated point.

Step Descriptions

- Random Start Point: In this project, for each functions, I started with randomly generated 30-dimension point.
- Generate Neighbor: The points are 30-dimensional. I generated neighbors by mutate 50% of the dimensions of the current point. The mutation is simply increase or decrease a small amount of value (within the range of (0, 1)) based on current point dimension.
- Fitness Calculation: The fitness functions are listed out as described and listed below. They are deterministic.

- Accept or Reject: Based on the comparison of fitness values between current point and mutated point, we accepts the mutation if the fitness is lower then current fitness value. Otherwise, we keep unchanged and try to find other neighbor which could be potentially better than current one.
- Update Current Loop Result: If we accept the mutation, we update the current point values of all the dimensions; otherwise, we keep the current point the same.

Detail Attention:

- Check the range of the mutated dimensions. Each mutated dimension must be within the range of function definition.
- At end of each mutation, after having updated decision results, we need to clean up the loop temporary values, like the vector of integer to record the mutated dimension indexes, the vector of double used to record mutated point.

The hill climbing algorithm pseudo-code for Schwefel function is pasted below:

```
Continuous Space Hill Climbing Algorithm
Loop until no changes happen any more
    Generate a random solution S
    loop do
        Fitness(S) = EVAL(S);
        Generate 15 random integer within [0, 29] used as mutated dimen
        Generate random double [0.0, 1.0] as changes added to those 15
        Calculate mutated point S'
␣␣␣␣␣␣␣␣Fitness(S') = EVAL(S');
␣␣␣␣␣␣␣␣if␣(␣Fitness(S') < Fitness(S) )
            S <-- S'
␣␣␣␣␣␣␣␣␣␣␣␣Fitness(S)␣<--␣Fitness(S')
End loop
return S;
```

1.2 Simulated Annealing

Simulated annealing is a generic meta-heuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space.

Compared with hill climbing, instead of always selecting the best neighbor, Simulated Annealing does the same accept the neighbor when the Neighbor is already performs better than current position. While Simulated Annealing also accepts worse neighbor with some probability when the currently mutated worse Neighbor has a good chance potentially leading to a better global optimum. This probability is a function of Annealing temperature. As the temperature reduced down, the probability of accepting worse Neighbor is also reduced down.

Since Simulated Annealing accepts worse Neighbors when necessary, and in the end of loop I may not have found the global optimum yet, I would better use a vector of double type to record the best point. And later on, I could restart from this best point.

Data Structure: I have used about the same data Structure as used in hill climbing.

- In this project, I have used double precision float variables to record mutated fitness and best fitness.
- I used a vector of integer of length 15 to record the mutated dimensions,
- a vector of double (of length 30) to record the best point that minimize the function, and
- another vector of double (of length 30) to record the mutated point.

Step Descriptions

- Random Start Point: In this project, for each functions, I started with randomly generated 30-dimension point.
- Generate Neighbor: The points are 30-dimensional. I generated neighbors by mutate 50% of the dimensions of the current point. The mutation is simply increase or decrease a small amount of value (within the range of (0, 1)) based on current point dimension.
- Fitness Calculation: The fitness functions are listed out as described and listed below. They are deterministic.
- Accept or Reject: Based on the comparison of fitness values between current point and mutated point, we accepts the mutation if the fitness is lower then current fitness value already. Otherwise, we accept changes only with certain probability when the mutated point has good chance leading to potentially better global optimum.
- Update Current Loop Result: If we accept the mutation, we update the current point values of all the dimensions; otherwise, we keep the current point the same.
- Temperature Schedule: The temperature is reduced by a constant floating factor near one.

Detail Attention:

- Check the range of the mutated dimensions. Each mutated dimension must be within the range of function definition.

- Since we may give up better current point Compared with mutated one, when we meet better solutions, we need to update the best fitness value and the best point vector.
- At end of each mutation, after having updated decision results, we need to clean up the loop temporary values, like the vector of integer to record the mutated dimension indexes, the vector of double used to record mutated point.

The simulated Annealing algorithm pseudo-code for Schwefel function is pasted below:

```
Start with a Random point S
for T = 100 to 0 step -1:
    Fitness(S) = EVAL(S)
    Pick a Neighbor of S, S'
    Fitness(S') = EVAL(S')
    if ( Fitness(S') < Fitness(S) )
        S <-- S'
    else
    With probability P( Fitness(S), Fitness(S'), T)
        S <-- S'
return S
```

2 Results

The hill climbing Algorithm supposed to find the global optimum, here listed the hill climbing results for both functions.

name	beginning Sphere	Ending Sphere	Beginning Schwefel	Ending Schwefel
x[0]	-4.5000	-4.5000	-412.0200	0.0000
x[1]	1.3000	1.3000	435.0000	435.0000
x[2]	0.4500	0.4500	47.0000	47.0000
x[3]	2.0550	2.0550	511.0000	511.0000
x[4]	2.0120	2.0120	176.0000	0.0000
x[5]	-3.5000	-4.0500	-112.0000	-112.0000
x[6]	2.3000	2.3000	235.0000	235.0000
x[7]	2.4500	2.4500	447.0000	447.0000
x[8]	-3.0550	-3.0550	509.0000	0.0000
x[9]	5.0120	-4.0500	476.0000	476.0000
x[10]	-2.5000	-2.5000	-512.0000	-512.0000
x[11]	3.3000	3.3000	135.0000	135.0000
x[12]	0.4500	0.4500	347.0000	347.0000
x[13]	-1.0550	-1.0550	209.0000	209.0000
x[14]	-5.0120	-5.0120	511.9600	511.9600
x[15]	-1.5000	-1.5000	-312.0000	-312.0000
x[16]	4.3000	4.3000	335.0000	335.0000
x[17]	0.4500	0.4500	147.0000	147.0000
x[18]	5.0550	5.0550	309.0000	309.0000
x[19]	4.0120	4.0120	376.0000	376.0000
x[20]	5.0500	5.0500	-412.0000	-412.0000
x[21]	-5.0000	-5.0000	435.0000	435.0000
x[22]	4.4500	4.4500	247.0000	247.0000
x[23]	1.0550	1.0550	509.0000	509.0000
x[24]	3.0120	3.0120	276.0000	276.0000
x[25]	-4.5000	-4.5000	311.0000	311.0000
x[26]	-5.1200	-5.1200	135.0000	135.0000
x[27]	-5.0500	-5.0500	397.0000	397.0000
x[28]	3.0550	3.0550	409.0000	409.0000
x[29]	-1.0120	-1.0120	476.0000	476.0000
Fitness	357.39091400	352.82577000	15885.60153515	15654.51787996

Table 1: Hill Climbing Algorithm Results

name	beginning Sphere	Ending Sphere	Beginning Schwefel	Ending Schwefel
x[0]	-4.5000	-4.5000	-412.0200	-412.0200
x[1]	1.3000	1.3000	435.0000	435.0000
x[2]	0.4500	0.4500	47.0000	47.0000
x[3]	2.0550	2.0550	511.0000	0.0000
x[4]	2.0120	2.0120	176.0000	0.0000
x[5]	-3.5000	-4.0500	-112.0000	-112.0000
x[6]	2.3000	2.3000	235.0000	235.0000
x[7]	2.4500	2.4500	447.0000	447.0000
x[8]	-3.0550	-3.0550	509.0000	0.0000
x[9]	5.0120	-4.0500	476.0000	0.0000
x[10]	-2.5000	-2.5000	-512.0000	-512.0041
x[11]	3.3000	3.3000	135.0000	135.0000
x[12]	0.4500	0.4500	347.0000	347.0000
x[13]	-1.0550	-1.0550	209.0000	209.0000
x[14]	-5.0120	-5.0120	511.9600	511.9681
x[15]	-1.5000	-1.4999	-312.0000	511.9681
x[16]	4.3000	4.3000	35.0000	335.0000
x[17]	0.4500	0.4500	147.0000	147.0000
x[18]	5.0550	5.0550	309.0000	309.0000
x[19]	4.0120	4.0120	376.0000	376.0000
x[20]	5.0500	5.0500	-412.0000	511.9681
x[21]	-5.0000	-5.0000	435.0000	435.0000
x[22]	4.4500	4.4500	247.0000	511.9681
x[23]	1.0550	1.0550	509.0000	509.0000
x[24]	3.0120	3.0115	276.0000	511.9681
x[25]	-4.5000	-4.5000	311.0000	311.0000
x[26]	-5.1200	-5.1200	135.0000	135.0000
x[27]	-5.0500	-5.0500	397.0000	397.0000
x[28]	3.0550	3.0550	409.0000	409.0000
x[29]	-1.0120	-1.0120	476.0000	476.0000
Fitness	357.39091400	350.82548000	15885.60153515	15546.72769305

Table 2: Simulated Annealing Algorithm Results

3 Conclusions

Hill climbing is supposed to find the local optimum, and Simulated Annealing should be able to find the global optimum. The results from both methods and both functions indicate that Simulated Annealing Algorithm does find better solutions than hill climbing. But due to time and parameter selection, the Simulated Annealing Algorithm has not been able to find the global optimum yet.

The possible reasons that Simulated Annealing failed to find the global optimum is that:

- I selected the Neighbors completely random, though with 15 dimensions slight changes. The randomness make lose the built ground. So to find the global optimum takes time.
- The completely randomness of selecting Neighbors also produces trouble. Though we have tried to restrict the range to be 15 dimensions each mutate point, and within [0, 1] changes to each dimension value. There are still very big searching space.
- For these two questions, since both of them are separable, one possible solution is to try to find the global optimum for the Simulated Annealing Algorithm separately for each dimension, and then combine the globally optimal dimension solutions together to get the global optimum.