##### Writeup Part

### Problem. 1

### Part a)

The plot for part a/b/c is attached here:



Please refer to the code for detailed implementation. Specifically, we selected: .

### Part b)

The plot of DDS is added above. We used the implementation provided.

We have the following suggestions:

1. Make the r value reduce during the evaluation. That is, we can make the r value a function of the current iteration. Effectively, we are reducing the amount of perturbation in later iterations. This is similar to the idea of temperature in simulated annealing, where the later iterations tends to be more “greedy”, fine-tuning the solution in a smaller window.
2. Cut the tails of the normal distribution. In the current implementation, the reflection method simply mirrors the values at the end points, and adds them back to the distribution. This might cause undesired distributions, as if our value is close to the edge, the peak of distribution might not be at our current value. We would instead recommend cutting the tails of the normal distribution. That is, if the value after adding perturbation exceeds the boundaries, we simply redo the perturbation until the value satisfies requirements. It can be shown that this method requires no more than 2 tries in expectation, and thus should be reasonably efficient.

### Part c)

The plot of GA is added above.

From the plot, it seems that DDS and SA perform relatively similarly, where SA performs slightly better over all function evaluations. Both DDS and SA seem to dominate GA from the plot.

### Part d)

The results for the 3 algorithms of the best function values per each trial:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Trial | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| DDS | 0.72513 | 0.70825 | 0.71831 | 0.74809 | 0.71433 | 0.76385 | 0.76094 |
| GA | 0.587382 | 0.575319 | 0.606831 | 0.590169 | 0.584489 | 0.588266 | 0.60199 |
| SA | 0.74575 | 0.741014 | 0.753614 | 0.755138 | 0.75369 | 0.771203 | 0.766349 |
| Trial | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| DDS | 0.69754 | 0.73678 | 0.74212 | 0.75094 | 0.71568 | 0.76212 | 0.75417 |
| GA | 0.577118 | 0.545315 | 0.449049 | 0.612161 | 0.600185 | 0.486874 | 0.56873 |
| SA | 0.772618 | 0.767772 | 0.750842 | 0.7588 | 0.772877 | 0.761527 | 0.728569 |
| Trial | 15 | 16 | 17 | 18 | 19 | 20 |  |
| DDS | 0.70109 | 0.70675 | 0.73881 | 0.70731 | 0.73868 | 0.75904 |  |
| GA | 0.500606 | 0.476849 | 0.55651 | 0.618526 | 0.629588 | 0.589811 |  |
| SA | 0.756162 | 0.752145 | 0.744717 | 0.767855 | 0.768977 | 0.688795 |  |

We used Boxplot, Empirical CDF plot, and Hypothesis testing to compare the three algorithms.

Boxplot:



From the Boxplot, we can clearly conclude that GA is being dominated by both algorithms. SA seems to perform generally better than DDS, but with an outlier worse than the worst DDS performance.

Empirical CDF:



From the Empirical CDF, we can again conclude that GA is dominated by both DDS and SA. Also we notice that SA generally performs better than DDS, with the exception for the lowest objective function value point(outlier marked in the boxplot too).

Hypothesis testing:

We run the 2 sample t test on DDS with SA, DDS with GA, and SA with GA:

DDS with SA:

## Problem. 2

For each generation (given fitness is known), we calculate next generation and calculate the fitness of that generation.

**1 processor:**

One processor needs to do all algorithmic calculations (50s) plus time for each offspring (. Note, there is no communication time.

Each generation: .

So, total time is

**20 processors:**

One processor needs to do all algorithmic calculations (50s) plus each of the 20 processors will need time for each offspring . There is communication time: each of the 20 processors will need to spend 10s communicating

Each generation:

So, total time is

**22 processors:**

Same as 20 processor, since you cannot break up the task into more than 20 subtasks.

Total time is

### Part a)

If wall clock time is .

### Part b)

Same thing since we cannot break up the task into more than 20 subtasks:

Wall clock time: 30000s, and speedup = 3.5

### Part c)

For 20 processors:

To get an efficiency of at least 0.8,

For 22 processors:

To get an efficiency of at least 0.8,

For 22 processors, even though it takes the same amount of time to finish the task, the efficiency decreased because there will be 2 processors that don’t have anything to do. In addition, needs to increase asymptotically to increase efficiency since it is impossible for efficiency to be 1 unless .