# Meta-heuristic Approaches to Solving Next Release Problem

**Akshay Kumar Chaluvadi**

Computer Science Department  
North Carolina State University  
Raleigh, NC, 27606, USA  
[achaluv@ncsu.edu](mailto:achaluv@ncsu.edu)

**Pritesh Ranjan**

Computer Science Department  
North Carolina State University  
Raleigh, NC, 27606, USA  
[pranjan@ncsu.edu](mailto:pranjan@ncsu.edu)

**Raghav Potluri**

Computer Science Department  
North Carolina State University  
Raleigh, NC, 27606, USA  
[rpotlur@ncsu.edu](mailto:rpotlur@ncsu.edu)

# 1. Abstract

Companies developing and maintaining software need to determine which feature should be added as a part of their next release. They wish to select the features to ensure the satisfaction of customer is high and cost of releasing the feature is within the allocated budget. This situation is modelled as a NP hard problem. The use of various modern optimizers to find a near optimal solution is theme of the project. We used Ant Colony Optimizer, Differential Evolution and Genetic Algorithm to compare the results against. Our results show how Differential Evolution outclassed the other optimizers in terms of results.

# 2. Introduction

To understand the Optimizers one should clearly understand the model being used. The Next Release problem can be described as: Given a software package, there is a set, C, of m customers whose requirements must be considered in the development of the next release of this system. There is also a set, R, of n requirements to complete. To meet each requirement, it is needed to expend a certain amount of resources, which can be transformed into an economical cost: the cost of satisfying the requirement. Additionally, different clients have different interests in the implementation of each requirement. The first component of the objective function expresses the weighted overall satisfaction of the stakeholders. The other component deals with risk management, expressing those requirements with higher risk should be implemented earlier. Finally, the two restrictions will, respectively, limit the implementation cost of requirements in each release and guarantee that the precedence among requirements will be respected.

In the following sections we will be discussing our approach to dealing with MO-NRP problem.  We have implemented Genetic algorithm, Differential Evolution and Ant Colony Optimization in our project work. In **section 3,** we describe the dataset we worked with. In **section 4** presents a note on two methods to determine Pareto Dominance – Binary Dominance (BDOM) and Continuous Dominance (CDOM). In **section 5**, we describe generating population and random solutions to the problem. **Section 6,** describes the GA algorithm. **Section 7**, describes the Differential Evolution approach that we employed. In **section 8,** we describe a way to **tune the ‘extrapolate amount’ variable** for DE using Simulated Annealing and comment on the values of generated for different initial population size for DE. We also present a comparison of the results (spread and Inter Generation Distance) with different values of . Finally, in **section 9** we describe an implementation of Ant Colony Optimization, and the results we obtained for NRP.

**3. The Dataset**:

We have generated artificial dataset for our NRP model. The parameter for the model were as follows:

*Requirements: Number of Requirements* = 30

The range of cost for requirements is from 1 to 200. A 200 means the requirement is associated with high risk & cost of implementation, while a 0 means that an intern can be assigned this job.

Requirements table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9 | r10 | r11 | r12 | r13 | r14 | r15 | r16 | r17 | r18 | r19 | r20 | r21 | r22 | r23 | r24 | r25 | r26 | r27 | r28 | r29 |
| 47 | 47 | 44 | 92 | 58 | 5 | 168 | 112 | 129 | 38 | 199 | 172 | 25 | 67 | 145 | 143 | 188 | 85 | 167 | 135 | 61 | 118 | 177 | 170 | 102 | 118 | 7 | 49 | 160 | 83 |

----------------------------------------------------------------------------------------------------------------------------------------------------------------

*Clients*: Number of clients

The range for clients is integer value from 1 to 10. A 10 means the software company will do anything to keep this client happy, while a 1 means, the company doesn’t mind loosing this client.

Clients table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 | c15 | c16 | c17 | c18 | c19 |
| 9 | 5 | 8 | 1 | 5 | 8 | 3 | 10 | 10 | 1 | 1 | 6 | 10 | 4 | 3 | 5 | 1 | 3 | 5 | 5 |

----------------------------------------------------------------------------------------------------------------------------------------------------------------

*Budget*: 1000

*The importance matrix*: Number of Clients by Number of Requirements. Each cell represents the importance a gave for a .

The range for importance is integer values from 0 to 100. A 100 means that the particular client desperately wants that feature, while 0 means the client doesn’t care.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9 | r10 | r11 | r12 | r13 | r14 | r15 | r16 | r17 | r18 | r19 | r20 | r21 | r22 | r23 | r24 | r25 | r26 | r27 | r28 | r29 |
| c0 | 17 | 55 | 71 | 68 | 37 | 44 | 51 | 78 | 52 | 39 | 49 | 2 | 4 | 71 | 99 | 59 | 39 | 17 | 50 | 99 | 77 | 54 | 86 | 23 | 51 | 96 | 58 | 46 | 27 | 55 |
| c1 | 96 | 0 | 79 | 82 | 89 | 74 | 81 | 52 | 56 | 43 | 5 | 87 | 57 | 20 | 50 | 48 | 36 | 34 | 54 | 62 | 61 | 46 | 2 | 23 | 17 | 59 | 86 | 80 | 80 | 82 |
| c2 | 25 | 85 | 67 | 8 | 1 | 1 | 76 | 25 | 11 | 63 | 34 | 7 | 16 | 53 | 16 | 27 | 71 | 45 | 32 | 47 | 2 | 39 | 42 | 18 | 10 | 90 | 51 | 21 | 61 | 82 |
| c3 | 2 | 1 | 14 | 72 | 16 | 71 | 68 | 55 | 22 | 98 | 80 | 52 | 22 | 65 | 39 | 58 | 32 | 63 | 5 | 30 | 97 | 88 | 30 | 86 | 31 | 94 | 75 | 42 | 25 | 0 |
| c4 | 88 | 3 | 82 | 97 | 57 | 17 | 87 | 98 | 71 | 51 | 38 | 35 | 20 | 68 | 43 | 19 | 10 | 67 | 29 | 50 | 32 | 88 | 90 | 1 | 20 | 33 | 99 | 79 | 34 | 21 |
| c5 | 68 | 84 | 94 | 34 | 89 | 69 | 48 | 99 | 23 | 73 | 8 | 17 | 92 | 21 | 76 | 60 | 84 | 37 | 34 | 29 | 87 | 61 | 96 | 89 | 13 | 55 | 10 | 3 | 7 | 87 |
| c6 | 79 | 83 | 34 | 62 | 78 | 38 | 57 | 22 | 8 | 26 | 89 | 57 | 93 | 46 | 27 | 79 | 83 | 1 | 67 | 9 | 11 | 89 | 4 | 24 | 99 | 42 | 11 | 16 | 24 | 75 |
| c7 | 10 | 91 | 38 | 97 | 91 | 29 | 25 | 48 | 10 | 65 | 4 | 1 | 99 | 29 | 60 | 45 | 31 | 6 | 92 | 97 | 97 | 11 | 21 | 62 | 98 | 54 | 69 | 66 | 26 | 54 |
| c8 | 31 | 24 | 8 | 28 | 99 | 45 | 65 | 64 | 95 | 39 | 30 | 33 | 31 | 85 | 90 | 30 | 33 | 54 | 58 | 60 | 24 | 2 | 24 | 7 | 55 | 7 | 7 | 64 | 29 | 80 |
| c9 | 49 | 87 | 15 | 50 | 80 | 7 | 95 | 17 | 78 | 99 | 82 | 32 | 10 | 51 | 92 | 29 | 90 | 14 | 91 | 3 | 31 | 91 | 81 | 91 | 84 | 75 | 69 | 17 | 43 | 15 |
| c10 | 72 | 67 | 25 | 6 | 97 | 81 | 55 | 54 | 85 | 45 | 39 | 34 | 26 | 2 | 65 | 42 | 57 | 6 | 35 | 13 | 12 | 26 | 83 | 40 | 40 | 61 | 23 | 0 | 53 | 50 |
| c11 | 65 | 44 | 69 | 73 | 24 | 50 | 48 | 22 | 41 | 56 | 91 | 92 | 27 | 65 | 4 | 7 | 51 | 88 | 16 | 77 | 89 | 31 | 69 | 85 | 37 | 70 | 74 | 60 | 86 | 90 |
| c12 | 96 | 57 | 17 | 25 | 21 | 57 | 76 | 5 | 68 | 72 | 35 | 52 | 16 | 73 | 4 | 99 | 81 | 63 | 27 | 92 | 96 | 14 | 78 | 85 | 66 | 70 | 44 | 93 | 98 | 38 |
| c13 | 81 | 43 | 16 | 32 | 12 | 91 | 96 | 12 | 60 | 41 | 11 | 29 | 25 | 75 | 0 | 19 | 44 | 2 | 63 | 61 | 84 | 20 | 28 | 54 | 27 | 59 | 25 | 69 | 79 | 81 |
| c14 | 98 | 55 | 49 | 86 | 77 | 57 | 38 | 28 | 10 | 81 | 11 | 75 | 55 | 97 | 76 | 98 | 13 | 50 | 57 | 31 | 50 | 36 | 53 | 0 | 44 | 45 | 30 | 40 | 79 | 69 |
| c15 | 49 | 65 | 38 | 20 | 0 | 28 | 60 | 89 | 83 | 51 | 99 | 46 | 84 | 41 | 75 | 99 | 30 | 17 | 62 | 53 | 36 | 0 | 39 | 43 | 40 | 86 | 59 | 74 | 90 | 75 |
| c16 | 49 | 75 | 64 | 65 | 63 | 41 | 63 | 64 | 94 | 79 | 85 | 77 | 82 | 61 | 35 | 26 | 71 | 88 | 54 | 15 | 84 | 48 | 47 | 4 | 51 | 75 | 42 | 35 | 66 | 1 |
| c17 | 51 | 95 | 69 | 40 | 69 | 61 | 21 | 20 | 89 | 27 | 7 | 83 | 52 | 37 | 51 | 74 | 17 | 65 | 72 | 82 | 27 | 61 | 23 | 56 | 17 | 79 | 87 | 33 | 22 | 97 |
| c18 | 71 | 85 | 3 | 90 | 62 | 31 | 43 | 76 | 79 | 19 | 63 | 16 | 98 | 44 | 92 | 73 | 61 | 26 | 53 | 14 | 13 | 72 | 36 | 75 | 24 | 72 | 72 | 30 | 10 | 40 |
| c19 | 49 | 10 | 18 | 5 | 60 | 89 | 21 | 3 | 71 | 82 | 97 | 61 | 34 | 84 | 11 | 69 | 9 | 40 | 49 | 38 | 17 | 23 | 82 | 46 | 58 | 21 | 72 | 33 | 59 | 91 |

----------------------------------------------------------------------------------------------------------------------------------------------------------------

Note: Different dataset can be generated with different seed to random function.

The objective was to maximize total client satisfaction: [1]

While minimizing the total cost. (Secret, we are maximizing negative of cost.) [1]

As it can be readily observed the two objectives compete with each other, improving on one naturally means making the other worse. Hence, our goal is to find a set of solution that are on the Pareto frontier and let the software company decide exactly which solution they would want to go with.

**4. Pareto Dominance – Binary Dominance and Continuous Dominance.**

In order to compare two candidate solutions, we evaluate the candidates and then compare the objective values to determine which candidate is better. But, in case of multi-objective problems, one candidate can perform better in one dimension while losing on other. In such cases, we can use the concept of *Binary Dominance* (BDOM). According to BDOM a solution dominates another solution , if solution ’s objectives are never worse than solution and at least one objective in solution is better than its counterpart in [2].

While BDOM works well when number of objectives is less – ideally 2 – it doesn’t tell how good is the solution. Continuous Dominance (CDOM) – defined in [3] – sums the total improvement of solution over other solutions.

We will be showing that with our problem – even with 2 objectives – using CDOM substantially outperformed BDOM. We adapted CDOM code from here [5].

**5. Generating candidates and population:**

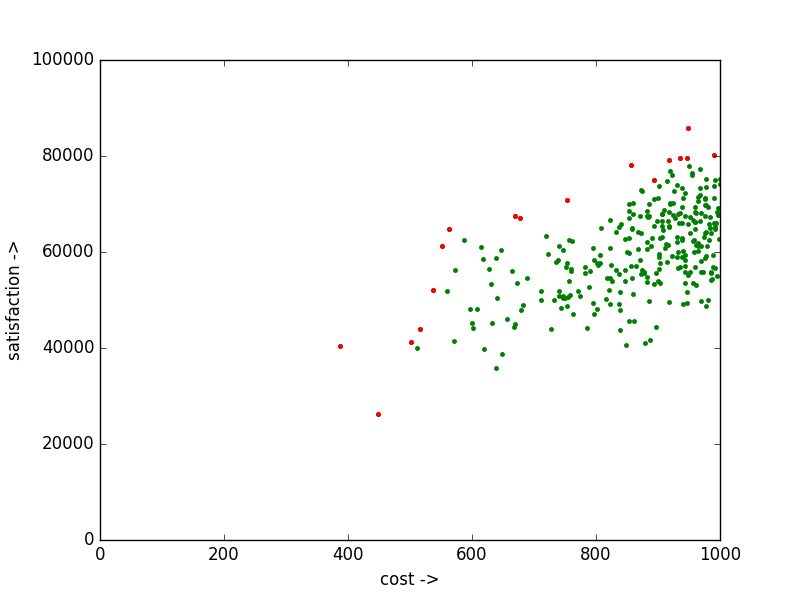
In evolutionary algorithms we need a candidate which is one of the possible solutions. Multiple such candidates put together is called a population. During subsequent runs of an evolutionary algorithm (called iteration) the quality of the population should increase. In our case a candidate is represented by a list of length equal to the number of requirements. The value in the list represents whether the requirement shall be implemented or not. A sample candidate:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9 | r10 | r11 | r12 | r13 | r14 | r15 | r16 | r17 | r18 | r19 | r20 | r21 | r22 | r23 | r24 | r25 | r26 | r27 | r28 | r29 |
| -1 | -1 | -1 | -1 | -1 | 0 | -1 | -1 | 0 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | -1 | -1 | -1 |

represents that the particular requirement should not be implemented, while means it should be implemented in this particular release.

To generate a population of given size, we first generate a candidate, check if the candidate satisfies all the constraints and add to the population set if it is not already there. We repeat this until the size of population set becomes equal to the specified population size.

The solutions generated (at random) in the previous step is a result of random search. This is the bottom-line result which must be outperformed by our meta-heuristic solutions. A plot of the objectives (cost – on X axis and Satisfaction – on Y axis) is shown in Figure 1.

****

**Figure 1: Initial population. Random solution. Red dots are the Pareto candidates among the initial population**

**6. Genetic Algorithm:**

We used the standard genetic algorithm, the pseudo code for the same is as follows:

Generate a population

While (stopping criteria):

Initialize a Children set.

For (number\_of\_iterations):

Mom, Dad = Pick two unique candidates from the population

X = Crossover(Mom, Dad)

Child = Mutate (X, mutation\_rate)

If Child satisfies the constraints:

Add Child to the Children set.

Union population set with the children set.

Do Elitism to pick most FIT candidates from population.

The different approaches we plugged in were:

**PopulationSize: 300**

**stopping criteria**: We ran the algorithm for a fixed number of generations. (specifically 10).

**number\_of\_iterations**: We ran the iteration for the population size, in a hope that each candidate will be picked at least once.

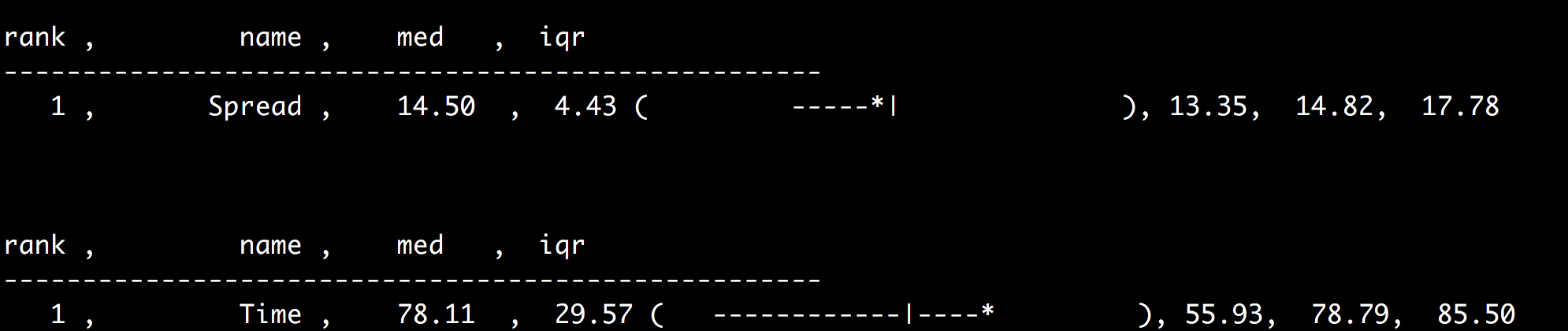
**Crossover**: We employed single point crossover, picking a character from Mom or Dad at random.

**Mutate and mutation\_rate**: For each character in child, change its value if a random number is less than mutation rate.

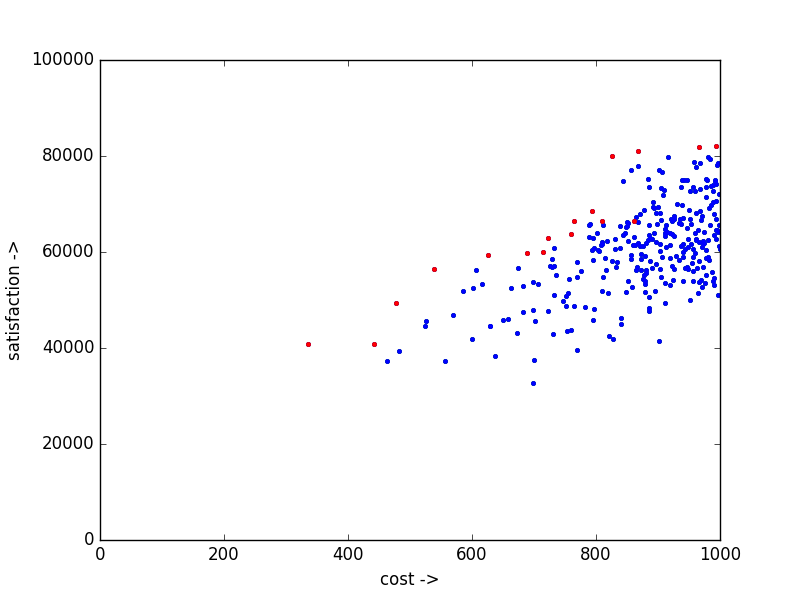
We observed that this step was the most important step in the GA process. Variations in mutation\_rate would generate candidates which were not valid. Initially we picked a mutation rate of 0.3 which would generate all invalid children. Picking mutation\_rate of 0.01, generated candidates out of whom half would be valid.

**Fitness:** We determined the fitness of a candidate as the number of other candidates in the population it dominated (using CDOM). For GA, using CDOM instead didn’t produce any significant improvement.

**Elitism**: Sort the candidates on the basis of decreasing Fitness and then pick the top candidates. The number of candidates we picked was the population size so that the population doesn’t increase in each generation.



**Figure 2: Mean, Median and IQR for Spread and time over 20 runs of GA**

A sample plot of objectives for the candidates after running the GA is shown in Figure 3:

**Figure 3: Population generated by GA. Blue dots represent the new candidates.**

**7. Differential Evolution:**

We followed the Differential Evolution algorithm as introduced in [4]. Our DE scheme was DE/RAND/1 for binomial parameters. Pseudo code for the specific algorithm that we implemented is as:

Generate a population

While (stopping criteria):

Pick 3 random unique candidates (A, B, C) from population.

Generate Extrapolate Vector F.

Generate the Child (T) by employing the DE scheme (DE/Rand/1)

If T satisfies the constraints:

If T dominates C and T:

Replace C by T

PopulationSize: 300

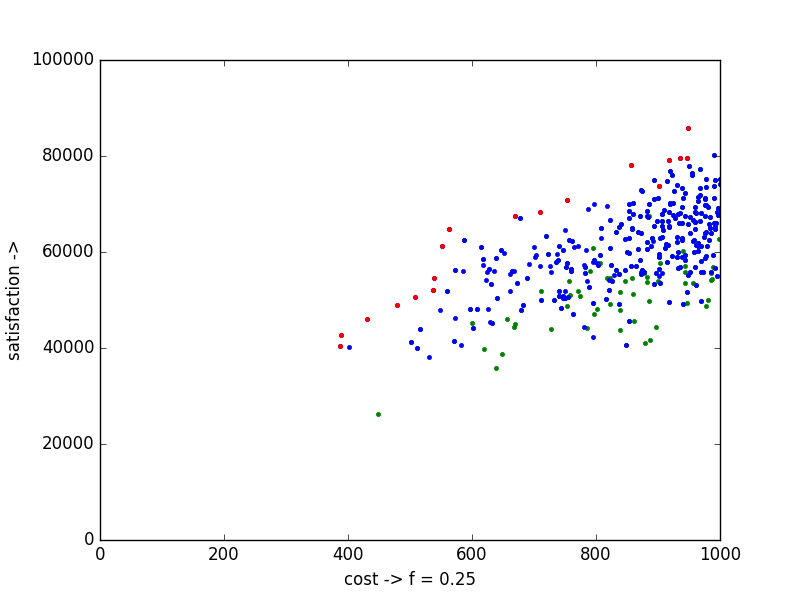
stopping criteria: We ran the algorithm for a fixed number of iterations. (specifically 2000). ***NOTE***: This means that DE ran for less number of iterations than GA. (For GA it was PopulationSize \* NumberOfGenerations i.e 300 \* 10.

Generate Extrapolate Vector: Generate a vector of length equal to number of requirements with binomial values. The value will be 0 if a random value (between 0 and 1) is less than extrapolate amount variable. ()

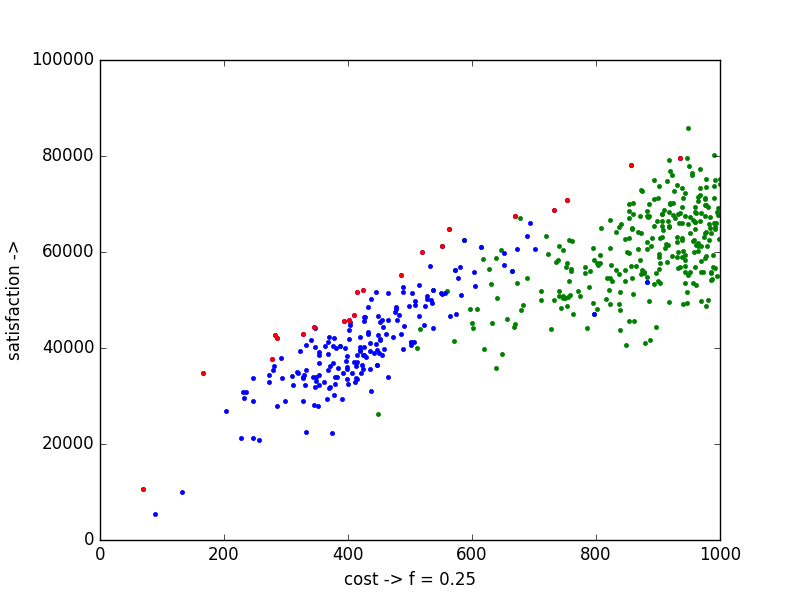
Generate the Child: In our case of binomial vectors we implemented the ‘DE/rand/1’ scheme as follows:

Pareto Dominance: We used BDOM and CDOM. The results with using CDOM were significantly better.

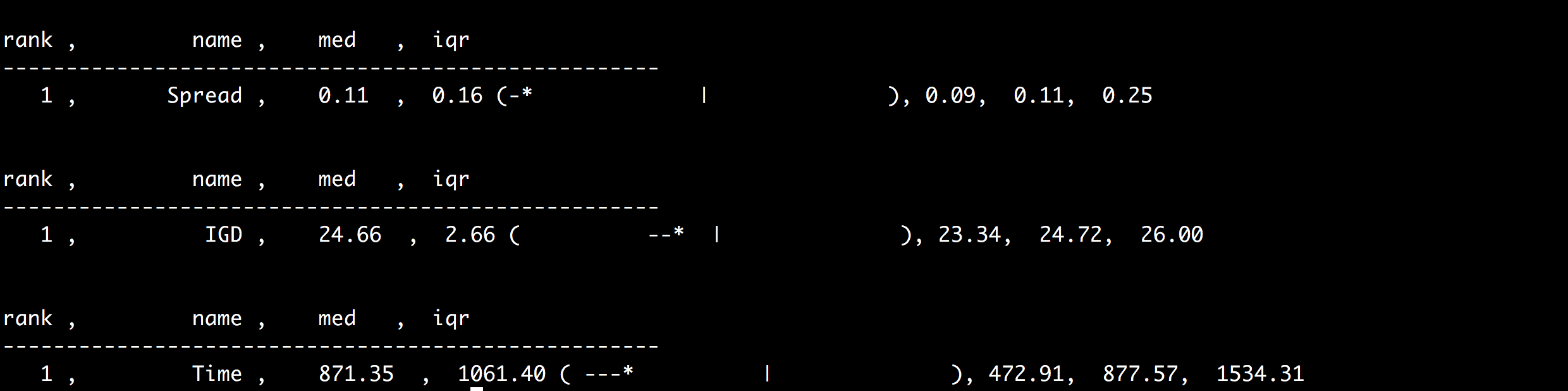
A sample plot of objectives for the candidates after running the DE with BDOM and CDOM is shown in figure 4 & 5 respectively.



**Figure 4: DE with and using BDOM**



**Figure 5: DE with and using CDOM**



**Figure 6:** **Mean, Median and IQR for Spread, IGD and time over 20 Runs of DE with CDOM**

**8. Parameter tuning for Extrapolate amount variable.**

*Decision:* We observed that varying had significant implications on performance of DE. Moreover, the same value of wouldn’t not suit for different population size and different dataset for NRP model. Hence, it seemed like a good idea to find an optimal value of before optimizing using DE.

*Objective:* For a particular value of the number of child vectors (T) generated that replaced the existing candidates. i.e. the number of ‘better candidates’.

*Optimizer:* Since this is single objective optimization problem, we selected simulated annealing as our optimizer. (plus, SA is simple, elegant, fast and cool).

The standard SA algorithm adapted with our energy and evaluate method:

number\_of\_iterations = 500

K\_MAX = 1000

best\_state = generate\_random\_f

best\_energy = current\_energy = normalize( evaluate( best\_state) ) )

k = 1

for number\_of\_iterations:

next\_f = generate\_random\_f

next\_energy = normalize( evaluate( best\_state) ) )

if next\_energy > best\_energy:

best\_energy = next\_energy

if next\_energy > current\_energy:

current\_energy = next\_energy

else if probability(old=current\_energy, new=next\_energy, k = k/K\_MAX) < random.random():

current\_enrgy = next\_energy

k += 1

Specifics of the methods used:

generate\_random\_f: generates a random value between 0 and 1

evaluate: For some run DE and report the number of ‘better candidates’ generated.

Normalize: Normalize the value reported by evaluate between 0 and 1

Probability(old, new, k): math.exp((new - old) / k)

NOTE: This is a maximization problem and the way I am using it needs probability() to return small values initially.

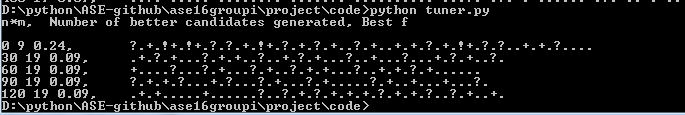
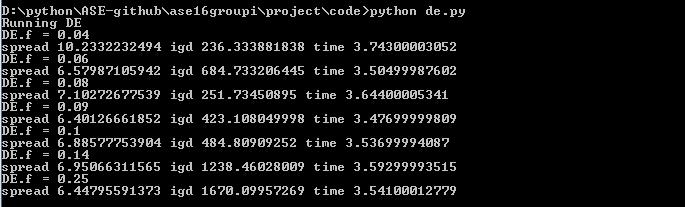


Figure 7: Sample run of hyper-parameter tuning module.

Next we ran DE for the following values of and the measured the spread and IGD for each f.

[0.04, 0.06, 0.08, 0.09, 0.1, 0.14, 0.25]



**Figure 8: Spread and IGD for DE with different value of Spread is the minimum for f = 0.09 while IGD is close to minimum.**

*Scope of improvement*: We observed that the values of f which give the best results are closer to each other. Hence, we believe that MAX-WALK-SAT could have performed better (during the local search phase) than SA.

Furthermore, as we have tuned for DE, we could have tuned mutation\_rate for GA and observed the results.

**9. Ant Colony Optimization:**

Our implementation of Ant Colony Optimizer was based on [6]. The paper was published for a model which has single objective. We enhanced by adding cost as another objective, making it multi-objective and used C-Dom to evaluate best solution.

We divided the implementation into generating data and running the optimizer upon the generated data. The algorithms we implemented uses C-Dom to evaluate between best choices. For D.E, we have implemented both C-Dom and B-Dom to evaluate the performance. Clearly, C-Dom performance was better. There are certain constants proposed by the paper [6], for ACO.

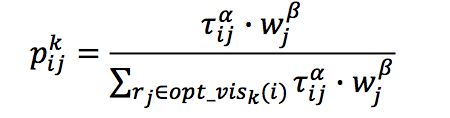
* ALPHA = 1
* BETA = 2
* Q = 1
* P = 0.1

## 9.1. Assumptions

* We assume that more than one customer can be concerned with any requirement, and that all the requirements are not equally important for all the customers.
* We assume that at most 20% of dependency between requirements.

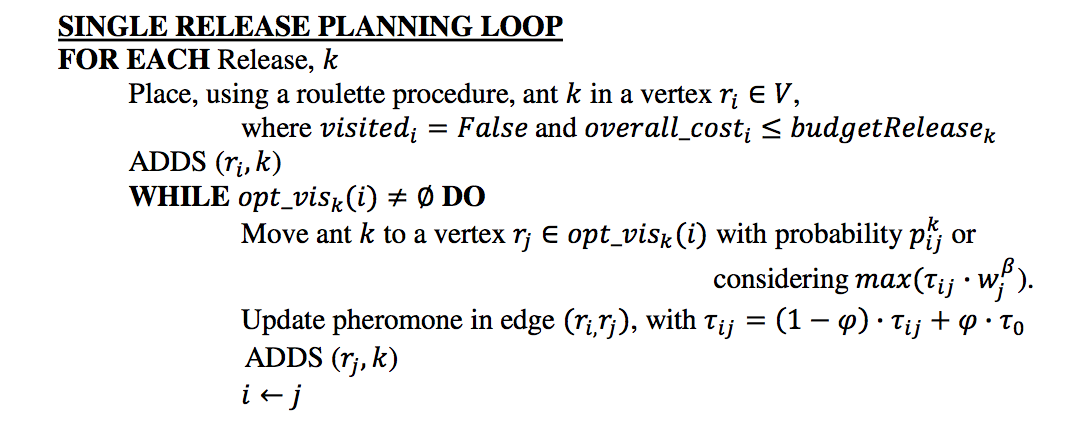
## 9.2 Implementation of Ant Colony Optimizer

A graph will be generated with the Release Planning problem instance information. Two types of edges will be created, representing mandatory and optional moves for an ant. Mandatory moves will be created to ensure that precedence constraints are respected. At each step, in case the random value q <= q0 (where q0 is a parameter), an ant visiting a node will have the probability of moving to the node is determined by [6]:



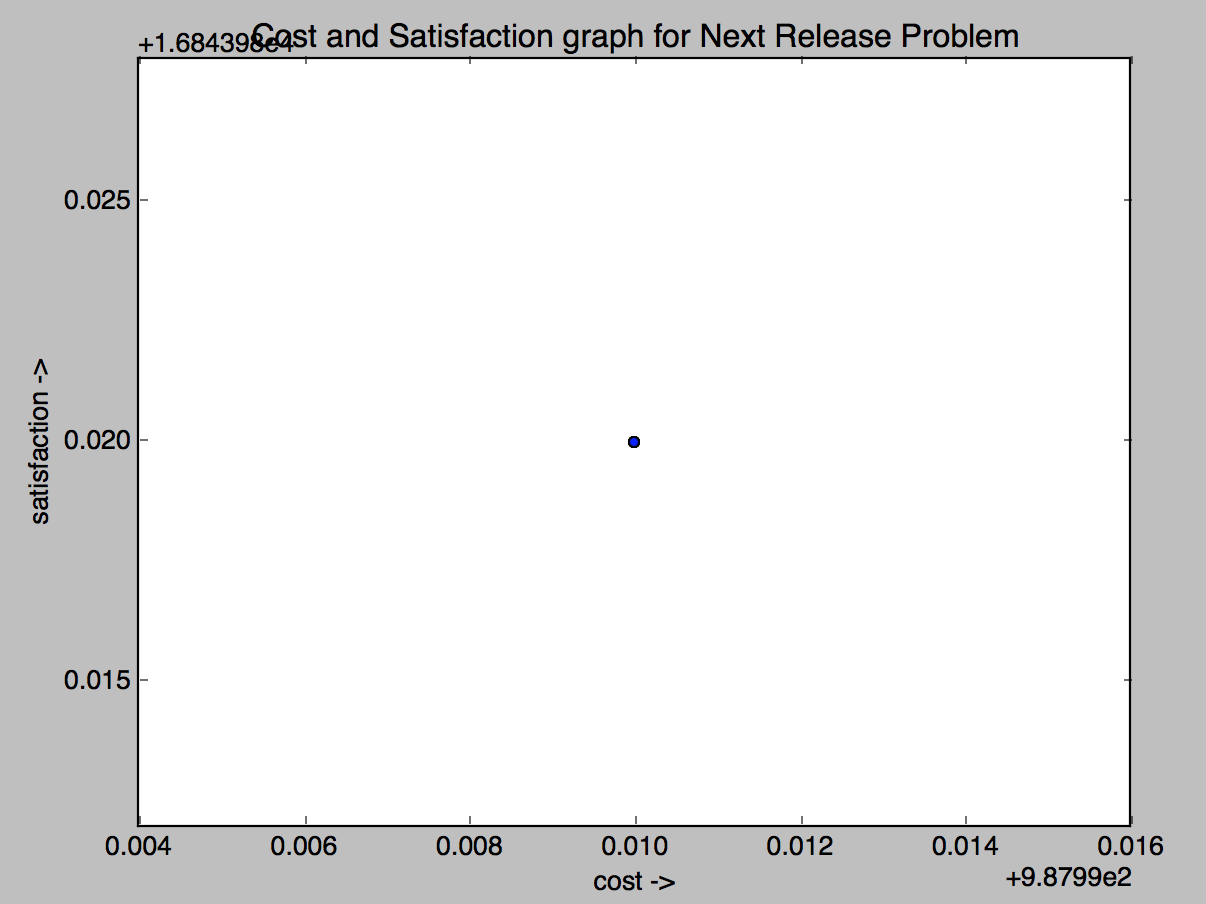
where “tau” is the amount of pheromone in edge and ALPHA, BETA are parameters controlling the relative importance of pheromone and requirement information.

A roulette procedure, considering the heuristic information measure of a node will be employed to select the initial node of each ant. The ant will travel through the generated graph, adding requirements to the first release until no more additions are allowed, due to budget constraints. The core algorithm is [6]:



**9.3. Results**

For given Requirements = 200, Customers = 10, Budget = 1000, we randomly generate client importance, client weight. The results are always between (977, 20780) indicating the cost, satisfaction coordinates. Below is a sample screenshot.



**Figure 9: The cost and satisfaction have been scaled down by 100000**

# 9.4 Conclusion

Based on our results, we can summarize Differential Evolution performs better than Ant Colony optimizer by:

* The number of results generated by D.E is very high compared to ACO.
* The best solution of ACO is usually out performed by majority of best solutions of D.E

## 10 Acknowledgement

We earnestly thank course instructor Dr. Tim Menzies, and teaching assistant George Mathew for giving us valuable advice in implementing the project.

**11. References:**

[1] J.J. Durillo et. al. “A Study of the Multi-Objective Next Release Problem.” At 1st International Symposium on Search Based Software Engineering

[2] J. Krall and T. Menzeis “GALE: Geometric Active Learning for Search-Based Software Engineering.”. IEEE Transactions on Software Engineering. VOL. 41.

[3] E. Zitzler et. al. “Indicator-based selection in multi-objective search,” in Proc. 8th Int. Conf. Parallel Problem Solving Nature, 2004, pp. 832–842.

[4] Storn, R. (1996). "On the usage of differential evolution for function optimization". Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS). pp. 519-523.

[5] [*https://github.com/ai-se/softgoals/blob/master/src/utilities/de.py#L141*](https://github.com/ai-se/softgoals/blob/master/src/utilities/de.py#L141)

[6] An Ant Colony Optimization Approach to the Software Release Planning with Dependent Requirements , Jerffeson Teixeira de Souza, Camila Loiola Brito Maia, Thiago do Nascimento Ferreira, Rafael Augusto Ferreira do Carmo, and Márcia Maria Albuquerque Brasil.