Human Resource Allocation Using Genetic Algorithm

Project Report
COMP 6776 Evolutionary Computing
Instructor: Dr. Hu
Prepared by: Aswini Naveen, Elham Karimi
April 2018

Abstract

With the latest advent in technology and organizational growth, recruiting interdisciplinary talents has become inevitable. As such interdisciplinary tech talents provide value and support for multiple parallel projects running across organization. They play a vital role especially in success of software projects compared to projects of other disciplines. This situation demands an efficient Human Resource Allocation (HRA) across an organization. In this work we address HRA using Genetic Algorithm. Our main goal is to provide optimal HRA in terms of minimizing the cost, duration and efficient workforce allocation, without compromising activity criticality. For this purpose, we adapt small data set, then we compare our model with one of similar search heuristic algorithm Simulated Annealing(SA). Both the algorithms(GA and SA) are implemented from scratch.

Key words: Human Resource Allocation, Genetic Algorithm, Simulated Annealing;

1. Introduction

HRA is an optimization problem and to solve resource optimization problems Search Based Software Engineering (SBSE) is used in various studies [2]. Our main idea behind this project is to take advantage of randomness that comes along naturally with Evolutionary Computation. Random resource allocation technique will benefit project managers significantly to limit the bottleneck situations they face, when a key resource employee resigns from the company. The three main advantages are i) Being able to provide on the job cross functional training to all level of employees ii) Knowledge transfer to employees belonging to a different area of expertise that facilitates professional growth and iii) Ensures opportunity to identify developer's area of interest and strength.

2. Problem Description

Software planning is becoming more complicated as the size of software project grows. As the number of phases in development life cycle increases resource allocation becomes more challenging and complicated. Old fashioned HRA models adopted by enterprises (i.e.., allocating one person on one type of work) would fail to maximize the personnel utilization. This situation demands an efficient HRA across an organization. Usually in software projects, Project Manager(PM) allocates resources and level resources using Resource Levelling Technique [2]. Unfortunately, Resource Leveling fails to ensure optimized resource allocation, project duration or the cost control as it is often done by manual adjustments.

2.1. Why GA in HRA?

Optimization of resource allocation is a NP-Hard problem. Genetic Algorithm can be a good candidate solver for optimization problems. GA does not guarantee best always but one of decent or good solution.

2.1.1. Possible Search Space (PSS)

PSS (1) = Ph1(n)+Ph1(n-1) +……. + Ph1(n-(n-1)), **PSS (2)** = Ph2(n)+Ph2(n-1) +……. + Ph2(n-(n-1)) ....
PSS(m) = Phm(n)+Phm(n-1) +……. + Phm(n-(n-1)), where Ph-Phase, m-Total number of phases, n-Total number of developers.

Total search space, PSS = PSS (1)*PSS (2)*….. *PSS(m)

In our sample model project implementation we have *7 developers ^11 phases = 1977326743 search space.*

3. Our GA Design

Our GA design guarantees to provide possible set of different best solutions that respects time, cost and developer allocation. Developer Allocation ensures a resource will be allocated to one task at a time to reduce multitasking time, as multitasking affects the productivity of a person and might result in erroneous work. A developer will be considered for next work only when he is done with his previously allocated task.

When our algorithm reaches convergence, the final generation's population will be filled with different set of possible solutions along with the best solution ever obtained across all generation. This gives the advantage for project managers to choose one of best solution that is suitable at that time.

In our project there are several challenges addressed while implementing HRA using Genetic Algorithm. The challenges are as follows:

- **Creating Appropriate Dataset** (like database structure) that will facilitate computing below respective fitness scores. We have created a python program to generate mock data set. However, to demonstrate our approach with legitimate dataset, we have used sample dataset from one of our reference paper [5] that is designed with different motto.
- **Computing Duration Fitness/Constraint** - Positive value if each phase/task completes before estimated time or else negative. If duration fitness is negative at least for one phase of project, then we discard that chromosome. *By this type of constraint in every phase, Duration Fitness is a Hard Constraint to satisfy.*
- **Computing Cost Fitness/Constraint** - Positive value if the overall project finishes within estimated budget or else negative. Even if the cost fitness is negative in a special phase and highly positive in other phase then the cost gets balanced for overall project. If the overall balanced project cost is negative, then it is evident that the project is a loss for the company. *By this type of constraint for overall project, Cost fitness is a Soft Constraint to satisfy.*
- **Computing Developer Allocation Fitness/Constraint**. This value should be equal to total number of phases. For every phase we cross check and ensure that the developers allocated to this task is not allocated to previous phase that is currently in progress. If any one of the task in the chromosome is allocated with a developer who is already working in previous task that is in progress, then we return zero immediately for that particular phase. Finally, if the cumulative sum of developer allocation fitness returned for every phase should be equal to total number of phases, meaning there is "No multitasking required for any developer". Any lesser value, indicate us that an unavailable developer is considered for this phase. *By this type of constraint in every phase, Developer Allocation Fitness is a Hard Constraint to satisfy.*
- **Consideration for Activity Criticality**. Activity Criticality is automatically satisfied by ensuring above Cost, Duration and Developer Allocation fitness.
- **Computing Overall Total Fitness for the Chromosome** - If any of the previous fitness constraint is not satisfied, then Total fitness becomes Zero, meaning the chromosome is replaceable in next generation. Greater than zero means the chromosome respects cost, duration, and developer allocation constraints listed above and is valid.

A chromosome that satisfies all the above computations and has total fitness value greater than zero, qualify as a solution candidate will be retained in the population for future generations. Our algorithm reaches convergence once the entire population gets filled up with valid chromosomes.

3.1. Representation

Below is our representation of chromosome as a complete solution for the overall project.

Where P is phase, d is developer, m is total number of phases and n is total number of developers. For example, in above chromosome for Phase_1, developer_1, developer_3 and developer_n are allocated and as such allocation continues until final phase of the project.

3.2. Initial Population

Ordered allocation of Phase followed by random number of *permutation-based allocation of developers* (using permutation to avoid same developer being assigned twice for one task).

3.3. Fitness Calculation

For all chromosomes in Initial Population provided the 3 constraints (cost, duration and allocation) are satisfied.
Overall Fitness = *Phase duration Fitness (Amount of time saved) + Cost Fitness (Profit of the project)+ Developer Allocation Fitness (Ideal cases, it is equal to total number of phases)*

If the constraints are not satisfied then,
Overall Fitness = **0**, meaning this is a replaceable chromosome in next generation.

3.4. Parent Selection

Satisfying all constraints and getting fitness value greater than zero rarely happens, so considering that any fitness-based selection

of parents will not a benefit for our GA. Thus, we have adapted, *Random parent selection*. However, we have strong survivor selection based on elitism. survivor selection will take care of filling the population with good solution over generations.

3.5. Variation

All the chromosome that has fitness value 0 will undergo, either Crossover or Mutation. Hence, the variation rate is 100% for invalid chromosomes.

Uniform Crossover is adapted, to avoid positional bias and to introduce some increasing variation range that will enable us to explore the search space.

Random Mutation is adapted, and all the phases developer set genes undergo mutation, when a chromosome is chosen to undergo mutation. We generate a developer set for every phase and cross check the new set with already existing set. Insert developers only if they are not already assigned for it. By doing this, there are chances that new developer set contains exactly same set of developer or less number of developer who are already assigned, this situation leads to inheriting parent's gene otherwise new set is assigned in children's gene.

3.6. Survivor Selection Elitism (μ, λ)

Chromosomes that got fitness zero will all be replaced with new chromosomes in next Generation. Chromosome that got fitness above zero, will be retained for further generation. Thus, when generation increases even the average fitness of population will improve and worst fitness will reach near to best fitness (convergence).

4. Our Model Project Implementation and Results

Our model project's Software Development Life Cycle (SDLC), consists of 11 phases and 7 developers.

There are eleven different phases such as, **Requirement Analysis Phase**(**P1**) followed by **Set of Design Phases**(**P2**: Front End, **P3**: Middleware, **P4**: Backend) followed by **Set of Implementation Phases**(**P5:** Front End, **P6**: Middleware, **P7**: Backend) followed by **Set of Testing Phases**(**P8:** Front End, **P9**: Middleware, **P10**: Backend) followed by final **Integrated Testing Phase**(**P11**). Below Fig.1, represents the idea mentioned above.

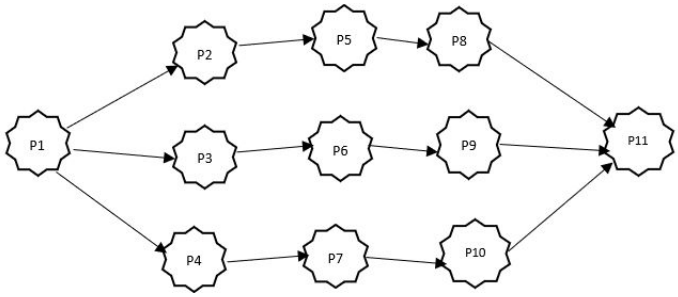


Fig 1. Our model project

4.1. Test Data

We assume that every company will have Developer Details and Project Details already in the database. With that idea, we have a sample data set that imitates two tables from database such as Developer Details and Project Details as follows.

4.1.1. Developer Details Dataset

Each developer has a skill level based on their experience and their respective abilities to contribute to a special type of task. For example, a systems architect Developer_1 can finish 1.25 quantum of Analysis work in a unit time, whereas an entry level Developer_6 can contribute only 0.75 quantum of work(Table I, dataset obtained from [4]).

Table I. Developer Details Dataset

Dev_ID	Dev_Level	Analysis	Design	Implementation	Testing
1	3	1.25	1	1.25	1.25
2	3	1.25	1	1.25	0.75
3	2	0.75	0.75	1	1
4	2	0.75	1	1	0.75
5	1	1	0.75	0.75	1
6	1	0.75	1	0.75	0.75
7	1	1	0.75	1	0.75

For an assumption and estimation, if the base pay is 15$/hr for an employee of the company, then resource level 3 charges 15*3/hr, level 2 charges 15*2/hr, level 1 charges 15*1/hr.

4.1.2. Project Cost Estimate

From above assumption, we can estimate maximum project budget, if all 3 levels of employee's work, on average (3*15+2*15+1*15)/3, *30$/hr*Total project duration* is ideal for budget calculation. In our case, project takes 1400 hrs of work, the maximum project cost would be 42,000$.

4.1.3. Project Details Dataset

Each phase in the project has minimum number of work hours required estimated during project estimation and updated in database. To emphasize on activity criticality, we have updated start hour and end hours based on phase dependency of that special phase.

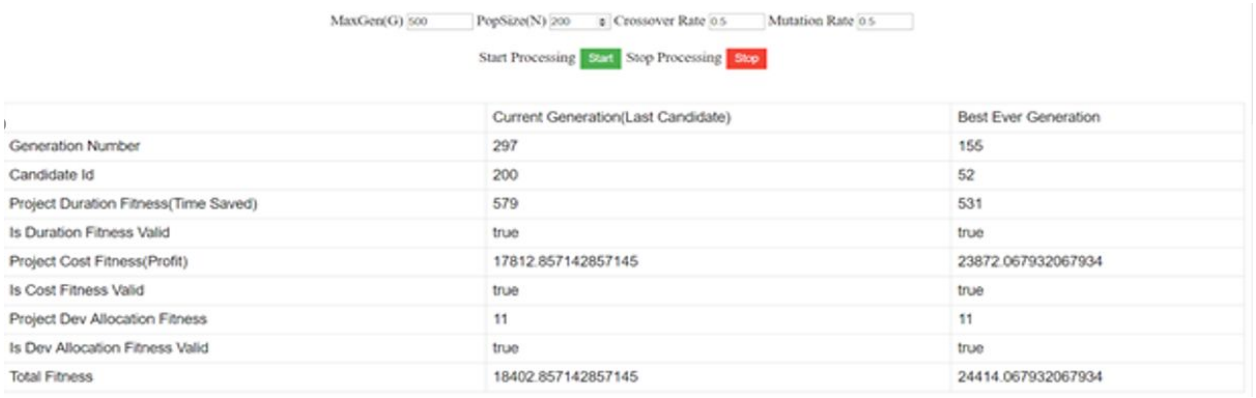
For example, analysis phase starts immediately as there is no dependency and completes at 400th hour. Any design phase (2,3,4) can begin only after requirement analysis phase (1), i.e.., Design phases Start at 400th hour after the wait is over for analysis phase. Similarly, we compute start and end hours based on phase dependency criteria. There is a special case in below table, where Integrated testing phase (10) depends up on all testing phase completion such as 8,9,10. For this type of multiple dependency phase we consider the maximum end time of dependencies. Max (1060,1320,870), 1320 would be the start time for Phase 11. Below is our sample dataset modified with extra fields from a reference paper.

Table II. Task Details Dataset

Phase_ ID	Phase_Type	Est_Duration(Hrs)	Start	End	Phase_dependency
1	Analysis	400	0	400	0
2	Design	320	400	720	1
3	Design	240	400	640	1
4	Design	240	400	640	1
5	Implementation	240	720	960	2
6	Implementation	600	640	1240	3
7	Implementation	160	640	200	4
8	Test	100	960	1060	5
9	Test	80	1240	1320	6
10	Test	70	800	870	7
11	Test	80	1320	1400	8,9,10

4.2. Our GA Result

We used JavaScript for implementation. Our GA fitness function is designed such that, higher the fitness better the solution is, as a Maximization Problem.



	Current Generation(Last Candidate)	Best Ever Generation
Generation Number	297	155
Candidate Id	200	52
Project Duration Fitness(Time Saved)	579	531
Is Duration Fitness Valid	true	true
Project Cost Fitness(Profit)	17812.857142857145	23872.067932067934
Is Cost Fitness Valid	true	true
Project Dev Allocation Fitness	11	11
Is Dev Allocation Fitness Valid	true	true
Total Fitness	18402.857142857145	24414.067932067934

Fig 2. Result of HRA using GA

We set population size as 200 and run our algorithm for 600 generations with crossover and mutation rate as 0.5 respectively. We pick last candidate from population pool as reference candidate (200th candidate) to compare with best candidate. Furthermore, we use this comparison to showcase how convergence happens with our strong survivor selection based on elitism (μ, λ).

From above screen shot we can see that Best Ever Generation was 155 and the candidate 52 was the best solution. We can also see that 200th candidate got one of best solution, from which point the convergence started(Can be seen clearly in third scenario graph), as we don't replace chromosomes with fitness value greater than 0.

We showcase two different scenarios with reference candidate over the evolution period, as follows

1. ***7th generation, 200th candidate*** has chromosome that leads to *32,614$ profit* and it also *satisfies project duration constraint*, however *failed to satisfy developer allocation*, it has only 6 so there are remaining 5 tasks that are overlapping with same developers.

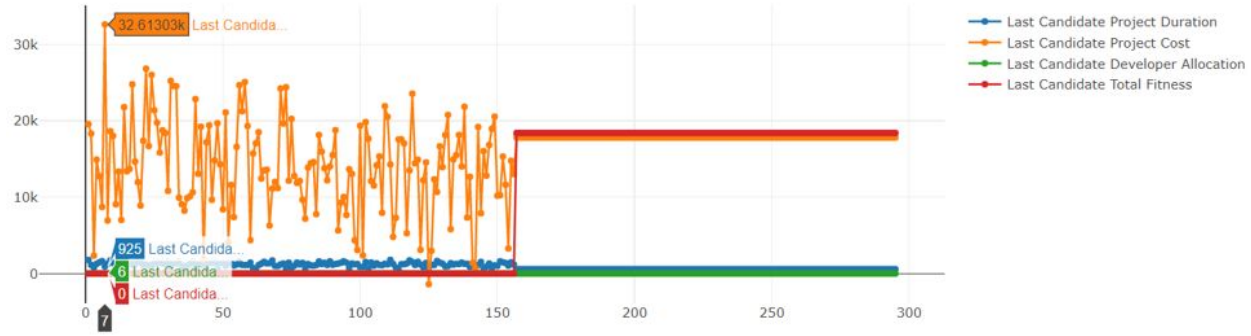


Fig 3. 7th generation, 200th candidate

2. ***157th generation, 200th candidate*** has chromosome that leads to *18,402$ profit, finishes 579 hours early satisfying project duration constraint*, and has *ideal developer allocation fitness 11 meaning "No multitasking for any developer"*. Convergence began at 157th generation onwards. Now 157th generation has 200 possible solutions.

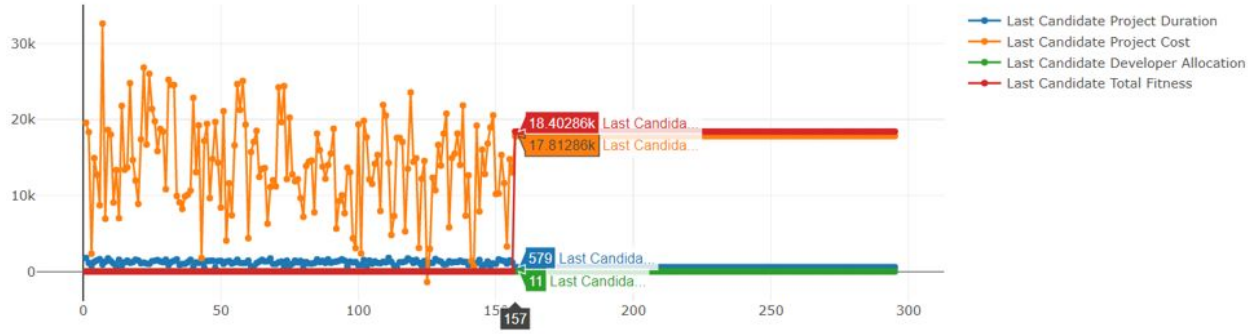


Fig 4. 157th generation, 200th candidate

Fig.5 presented below to showcase the connection between time taken from best ever solution is reached and last candidate getting best solution leading to convergence.

1. Fig.5 shows that 42nd generation had first best solution with 10,982 as overall fitness, at the same time 200th candidate (last candidate) had Zero fitness, meaning it had a replaceable chromosome.

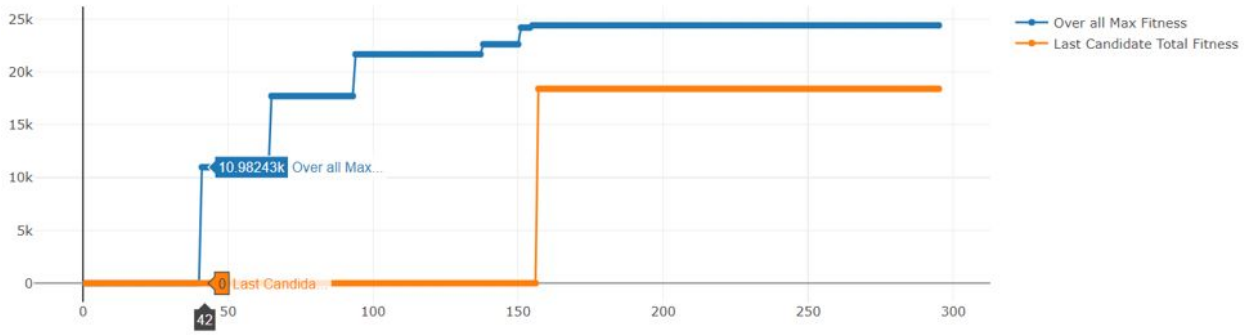


Fig 5. Last candidate reference vs First best solution at Gen 42

2. Fig.6 shows that at 157th generation 200th candidate had one of the best solution with 18,402 fitness. At the same time, we already got our overall best fit chromosome at 155th generation's 57th candidate with 24,410 fitness [can be cross verified in figure 1. Table representation]. After best solution is obtained in 42nd generation, it took 115 generations for our algorithm to get converged. Our design, continues the search for good solution until last candidate of population gets filled with good solution.

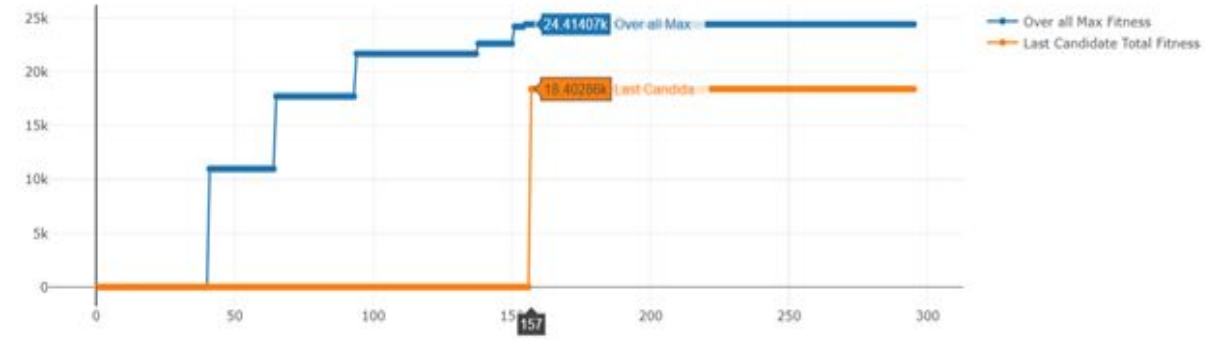


Fig 6. Last candidate's good soln., vs Optimum best solution leading to convergence at Gen 157.

5. Comparison and Discussion

We designed a similar Search Based Software Engineering (SBSE) algorithm, Simulated Annealing (SA).The name and inspiration behind SA, come from annealing in metallurgy, a technique involving heating and controlled cooling of a material[4]. The idea is to use temperature as a parameter to search through the space and identify best solution. Initially when temperature is high selection pressure is very low, that leads to replacing current solution with next solution even if next solution is not better than the current solution. At later stage, as temperature decreases, selection pressure is very high, very less likely a poor candidate will replace current candidate.

In our Genetic Algorithm design, we evaluate 200 candidates every generation and henceforth with 600 generation we evaluate 600*200 = 120,000 candidates overall(some may be same) to find best solution. We were able to find good solution most of the time when we ran our algorithm. We have attached screenshots and explanation before based on one of our best run. From above screenshots and below log info, we infer that by 155th generation we achieved the maximum and by 157th generation once 200th candidate got one of good solution our algorithm converged.

Best Fit Chromosome at *Gen **41*** with *Fitness **10,982***
0,[3, 2] | 1,[1] | 2,[3] | 3,[5, 6] | 4,[2, 5] | 5,[3, 6] | 6,[0] | 7,[5, 0, 2, 1] | 8,[0, 6] | 9,[0] | 10,[2, 3]
Best Fit Chromosome at Gen **65** with *Fitness **17.718***
0,[4, 5, 2] | 1,[1] | 2,[3] | 3,[5, 6] | 4,[2, 5] | 5,[3, 6] | 6,[0] | 7,[5, 0, 2, 1] | 8,[0, 6] | 9,[0] | 10,[2, 5, 3, 1, 4]
Best Fit Chromosome at *Gen **94*** with *Fitness **21.671***
0,[4, 5, 2] | 1,[1] | 2,[3] | 3,[5, 6] | 4,[2, 5] | 5,[6] | 6,[0] | 7,[4, 1] | 8,[0, 1, 4] | 9,[0] | 10,[2, 5, 3, 1, 4]
Best Fit Chromosome at *Gen **138*** with Fitness **22.620**

0,[2, 4, 6] | 1,[1] | 2,[3] | 3,[5, 2] | 4,[3] | 5,[6] | 6,[0] | 7,[4, 2, 0] | 8,[0, 6] | 9,[4] | 10,[2, 4]
Best Fit Chromosome at *Gen **151*** with *Fitness **24.197***
0,[4, 5, 2] | 1,[1, 3] | 2,[4, 6] | 3,[5, 2] | 4,[2, 5] | 5,[6] | 6,[0] | 7,[1, 5, 0] | 8,[0, 1, 4] | 9,[0] | 10,[5, 0, 3]
Best Fit Chromosome at *Gen **155*** with *Fitness **24.414**, saves **23872$**, saves **531 hours**, **11 No Overlap** of Allocation*
0,[2, 4, 6] | 1,[1] | 2,[3] | 3,[5, 6] | 4,[3] | 5,[6] | 6,[0] | 7,[4, 2, 0] | 8,[0, 6] | 9,[4] | 10,[5, 2]

Similarly for Simulated Annealing (SA), we set up 3200°celsius as temperature, 0.999999 as Alpha (α) that will facilitate to evaluate over 100,000 candidates. Every iteration a solution candidate is evaluated, so 100,000 iterations evaluate 100,000 candidates. From the below graph we can see that even after extending our evaluation for 128,801 iterations, we did not find a good solution. The search is still on as we can see the temperature(approx.,880.05°c) is still very high compared to our termination condition epsilon (0.001).

Fig 7. Simulated Annealing - Over 120,000 evaluations (no good solution).

We expanded our evaluation further(until 308,018 iterations), to verify if atleast before the temperature runs out, SA will find one good solution. We got one good solution at 157,806'th iteration as shown below. We cross verify from below table that 157,806'th iteration leads to *22,328$ profit, finishes 833 hours early satisfying project duration constraint*, *satisfying developer allocation*. The *'Red Spike'* in below graph indicates the 'One Good Solution' with fitness 23,172.6, rest all solutions turned out to be invalid ones even until 300,000 iterations.

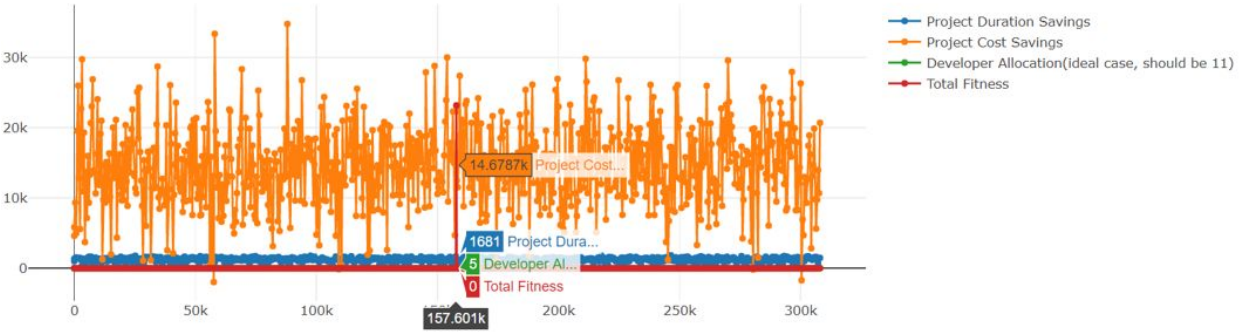


Fig 8. Simulated Annealing - Over 300,000 evaluations, one good solution (indicated with Red Spike)

Even though we gave advantage for SA by extending the time, our GA solution lead to *23,872$ profit whereas SA lead to 22,328$ profit.*

	Current Iteration	Good Solution
Temperature	147.04088653502268	660.3884340844313
Iteration Number	308018	157806
Project Duration Fitness(Time Saved)	1703	833
Is Duration Fitness Valid	true	true
Project Cost Fitness(Profit)	16932.825244930515	22328.61236802413
Is Cost Fitness Valid	true	true
Project Dev Allocation Fitness(	6	11
Is Dev Allocation Fitness Valid	false	true
Total Fitness	0	23172.61236802413

Fig 9. Results of HRA using Simulated Annealing

Overall, we achieved one good solution in SA after extending our search until 150,601 iterations. Whereas in GA, we were able to get best solution at 42nd generation thats is after evaluating just about 42*200 = 8400 candidates. In GA, 90% of the time we ran our algorithm for 600 generations, we were able to find good solution. This comparison shows us very clearly that GA, provides wider variation to explore search space provided with right parameter setting. In our case 200 pop., size, 100% variation for invalid chromosomes by either uniform crossover or random mutation enabled us to reach right search space. GA promises to converge once best solutions keeps filling in the population that is showcased in Fig.6.

6. Conclusion

We provided an optimum HRA using Genetic Algorithm. This algorithm can be used for multi-project planning. For example, by updating database with the estimated work hours for every individual phase with their dependency tasks for all the projects, we can derive start hours and end hours for every projects phase. Then, the algorithm can be run for all the projects together for available employee's database. GA guarantees best allocation for entire organization's project allocation with various possibilities and choices.

Due to dataset, time and computational power limitation we have demonstrated allocation for a small project with search space of approx., 197,732,674,3 and achieved result in less than 20 seconds approx., With the same or similar implementation, we understand that it is possible to derive efficient allocation results for fairly large sized projects and for multiple projects running across organizations as future work.

References

[1] J. Ai, "Genetic Algorithms-Based Model for Multi-Project Human Resource Allocation", Revista de la Facultad de Ingenier, 2017.

[2] N. Bibi, A. Ahsan and Z. Anwar "Project Resource Allocation Optimization using Search Based Software Engineering", IEEE, 2014.

[3] R. P. F. Abel, D. Kohlsdorf (2006). ACM recsys challenge 2016: Training data.

[4] J. Park Et al., "Practical Human Resource Allocation in Software Projects Using Genetic Algorithm".

[5] https://en.wikipedia.org/wiki/Simulated_annealing.