Software Next Release Planning Approach through Exact Optimization

Fabrício G. Freitas, Daniel P. Coutinho, Jerffeson T. Souza
Optimization in Software Engineering Group (GOES)
Natural and Intelligent Computation Laboratory (LACONI)
State University of Ceará (UECE)
Avenue Paranjana, 1700, Fortaleza, Brazil

ABSTRACT
The Software Requirements phase has notable importance, since it is responsible for the definition of the system itself. Several customers indicate which functionalities they want to be present in the software. However, constraints, such as budget, make it impossible to implement all desired requirements at once. One activity in this context is the release planning. The selection of which requirements should be implemented to the next release is necessary. In literature, metaheuristics have been employed to solve this problem. The objective of this work is to propose the use of exact optimization techniques in the problem, with the advantage that the resolution through these techniques ensures the best solutions. The results in several experiments show the validity of such application, in comparison with the metaheuristics approach.

General Terms
Exact Optimization, Software Requirements, Software Engineering.

Keywords
Search Based Software Engineering, Next Release Planning, Software Requirements.

1. INTRODUCTION
The software engineering plays an important role in the development of quality systems. Through decades of research, models and methodologies have been defined in order to support the software development process [1], considering that the final product quality is strongly related to the quality of the development process [2].

Unfortunately, such methodologies may not be appropriate to solve some software development problems, mainly in inherently complex problems. In those cases, automated methods should be used in order to solve the problems efficiently.

One important area in the software development process is the requirements engineering. This phase contains problems of high complexity, such as the Next Release Problem (NRP) [3]. This problem concerns on defining which requirements should be implemented for the next version of the system, according to customer satisfaction and budget constraints. Metaheuristics have been used to solve the problem, and so far the definition of the best solutions could not be guaranteed.

This paper proposes the resolution of the problem through exact techniques. We aim to find better solutions to the software development process, which is crucial to the area [4]. The research questions to be investigated are:

• Exact Optimization Applicability: Can exact techniques be applied to the problems?

• Exact Optimization Efficiency: Is exact optimization execution time an issue?

In order to answer those questions, we perform both effectiveness and efficiency comparison among exact techniques and metaheuristics to some instances of the problem. The instances were set in different sizes, in order to represent various contexts of application.

2. RELATED WORK
The Next Release Problem (NRP) was originally considered in 2001 using as objective function the maximization of customers' satisfaction [3]. The authors applied optimization techniques in five instances of the problem, and they considered three budget constraint scenarios: 30%, 50% and 70% of the total cost of all requirements. The authors employed the metaheuristics Simulated Annealing, Hill-Climbing, and a greedy technique. In the experiments it was found that Simulated Annealing technique yielded better results in comparison with other techniques.

In contrast to the previous related work, the main contribution of this work is to model and solve the NRP through exact optimization technique. Such an approach encourages the potential use of exact techniques on other problems of Software Engineering. It also plays as a contribution the reinforcement of the use of Operations Research techniques in Software Engineering contexts, indicating a cross-disciplinary approach.

The NRP was revisited in 2007, and a multiobjective formulation of maximization of customers' satisfaction and minimization of the implementing cost was taken [5]. In the paper, the customers' satisfaction function considers not only the importance of each client, but also the importance level that each customer has for each requirement. Multiobjective metaheuristic NSGA-II was able to solve the problem, though it could not guarantee the definition of the best set of solutions in the instances used.

3. SEARCH BASED SOFTWARE ENGINEERING
Software Engineering, as an engineering discipline, is a field with mathematical aspects and problems [6]. Additionally, as any engineering field, there are scenarios to optimize. An efficient way to solve this kind of problem, usually with

structural complexity and constraints, is an automated optimization approach [7].

The first step in this direction in software engineering was in 1976, where the problem of generating test data was attacked by numerical maximization [8]. Since 2001, however, this new approach to software engineering, know as Search based Software Engineering (SBSE), researchers and experts have intensified the modeling of Software Engineering problems as optimization problems [9].

The requirements engineering is a stage in which the software itself is defined, and therefore the activities performed at this point have an impact on all phases of development. Due to budget constraint, it is usually not possible to implement all the desired features. Thus, the selection of which requirements must be implemented consists of a relevant task in this phase. This problem can be tackled as an optimization problem.

The mathematical optimization problems are those with functions to be maximized or minimized and defined from coefficients and variables. The variables that define the functions may be subject to restrictions, i.e., the variables must satisfy a set of defined equations according to each instance of the problem. Formally, the optimization problem is defined as:

$$Minimize\ \{f_1(x),\ f_2(x),...,\ f_k(x)\}$$

$$Subject\ \ to:$$

$$g_1(x) \le 0, \quad i = \{i...p\}$$

$$h_1(x) \le 0, \quad i = \{i...q\}$$

In the mono-objective optimization, the search for solutions is performed according to the values of only one function. Thus, in a minimization problem, for example, if we take a solution A with function value less than the value of a solution B, then A is better than B.

3.1 Problem Definition

The mono-objective requirements selection [3] is defined by the following aspects:

- **Customers**: Consider a set $C = \{c_1, c_2, ... , c_m\}$ with m customers. Each client i has an importance value w_i. Each client indicates a list with the requirements desired.
- **Requirements**: Given a set $R = \{r_1, r_2, ... , r_n\}$ with n requirements. Each requirement j has an implementation cost $cost_j$, such as man-hours. The requirements also have a precedence relation, since some requirements depend on the implementation of others.
- **Company**: The development company has a maximum budget available B for use by the next version.

The mathematical formulation of mono-objective NRP is as follows:

$$\max \sum_{i=1}^{m} w_i \cdot X_i \qquad (1)$$

Subject to

$$\sum_{j=1}^{n} cost_j \cdot Y_j \le B \qquad (2)$$

$$Y_j \le Y_i, \qquad \forall r_j \ \ depends\ on\ r_i$$

The Boolean variable X_i indicates the selection state of customer i. If customer c_i is fully satisfied (all his desired requirements are selected), then X_i is 1. As shown in (1), the requirements selection considers maximizing the sum of importance of such customers. The Boolean variable Y_j represents the selection state of requirement j: Y_i is 1 if the requirement i is selected, and 0 otherwise. In (2), the restriction states that the selection is limited by the budget B.

The restriction (3) indicates the dependence relation among the requirements. Consider that some requirement r_j depends on other r_i. Therefore, in order to r_j be selected ($Y_i = 1$), X_i has to be greater or equal than 1, which in the binary context means that it is 1, i.e., requirement r_i is also selected in the solution.

The Figure 1 presents an example of the NRP structure, with 20 requirements (r_1.to r_{20}) and 10 customers (other c_1 to c_{10}). The links between requirements are the dependency relation.

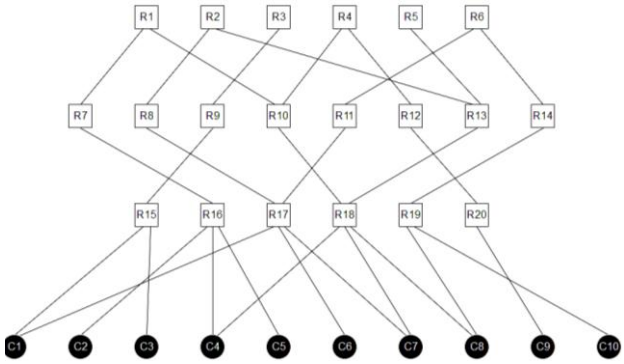

Fig 1: NRP illustrative example.

4. METHODS

4.1 Metaheuristics

The term metaheuristic [10] represents a class of generic search algorithms. These methods use ideas from various fields as inspiration to the process of trying to solve optimization problems. Metaheuristics approaches attempt to solve the problem by intelligently visiting only some solutions, but there is no guarantee that the best solution is returned. The following are summary on the mono-objective metaheuristics used.

- **Simulated Annealing**: The metaheuristic Simulated Annealing [11] is based on a physical process that occurs in the metals metallurgy. In the tempering process, a material is heated to high temperatures and, thereafter, is cooled so that at the end of the process the material is crystallized in a state of minimal energy. The relation with the mathematical optimization is the minimization of the objective function

value from the solutions found during the search process. The Simulated Annealing algorithm allows the acceptance of solutions that will not improve the value of the function. These acceptances against the objective are controlled by a statistical equation defined in the actual physical process. In summary, the algorithm works as follows: if the new solution is better than the current solution, then it is accepted, otherwise, if the new solution worsens the objective, then it is accepted with a certain probability defined in terms of difference between the solutions, the current value of the variable temperature and constant physical. Figure 2 next simulates how the Simulated Annealing works in a minimization problem.

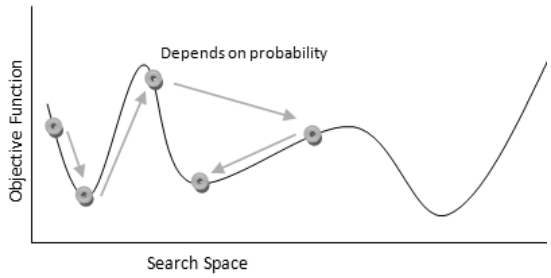

Fig 2: Simulated Annealing search procedure.

- **Genetic Algorithms**: The metaheuristic Genetic Algorithms [12] uses concepts of genetics, such as population and mutation. The metaheuristic can be defined in terms of two genetic operations: crossover, in which the structural information of two solutions are crossed to generate two new solutions; and the mutation process by which some random changes may occur in the solutions generated. The mutation is used in order to avoid the generation of the same solutions, and therefore to explore various search spaces. In summary, the current solutions are evaluated to determine which will continue to the next iteration ("generation"). Thus, the solutions are continuously selected according to performance in relation to the objective function. As new solutions are generated from these solutions that have been selected, the process "evolve" in order to generate better solutions.

4.2 Exact Optimization

The exact techniques are methods that use mathematical operations in order to solve the problem. The most known method for problems with linear both functions and constraints is the simplex method [13]. This method uses the formulation of the problem in matrix form, and applies matrix mathematical operations to achieve the global optimum, if any. The method is based on the geometric representation of the optimization problem in which the linear equations form a "polytope" in the search space. According to theorems in the field of linear programming, the optimal solution of the optimization problem is found in one of the vertices of the polytope considered as the system of linear equations of the formulation. Thus, from a valid initial solution, usually at the origin of space, the simplex method visits the adjacent vertices in the search for better values for the objective function. This approach is also based on the argument that indicates that the number of vertices of the polytope to a linear optimization problem is a finite number, and so the search will eventually end.

The process of visiting the adjacent vertices until it finds the global optimum is achieved by matrix operations including the exchange between the basic variables of the system of equations. This is accomplished by manipulation of the equations of the linear system that represents the model. In this work, the version used is the revised Simplex in the product form of inverse. For solving Integer Programming, the method used is Branch-and-Bound. This approach first solves the problem by using linear methods or nonlinear (depending on the problem treated) without the restriction of the integer solution in order to find the limits (bounds) of the solution. Additionally, the problem is divided into sub-problems in a tree structure (branch) in accordance with divisions of the domains of variables, and the divisions that are worse than the limit are discarded. The process is performed until the sub-problems have been considered.

5. METHODOLOGY

5.1 Instances

We generated random data sets for the simulation of various contexts. Although they are artificial data, this does not affect the present analysis, given that the data represent a random simulation of a context where the problems can happen, and thus serve as a basis for simulation. For the mono-objective Next Release Problem (NRP), we generated five instances of different sizes.

For the definition of the values domain, we followed the directions suggested in the original article that defined the problem [3]:

• Customers' value (w): 1 to 10
• Requirements cost: depends on the level;
  – 1 to 5 for the basic requirements,
  – 2 to 8 for second level,
  – 5 to 10 for third level.
• Number of basic requirements by customer: 1 to 2.

Table 1 shows the characteristics of each instance. Figure 1 shows the structure of instance

Table 1. Instances used in experiments.

Instance	Requirements	Customers
NRP-A	20	10
NRP-B	40	20
NRP-C	100	50
NRP-D	140	70
NRP-E	200	100

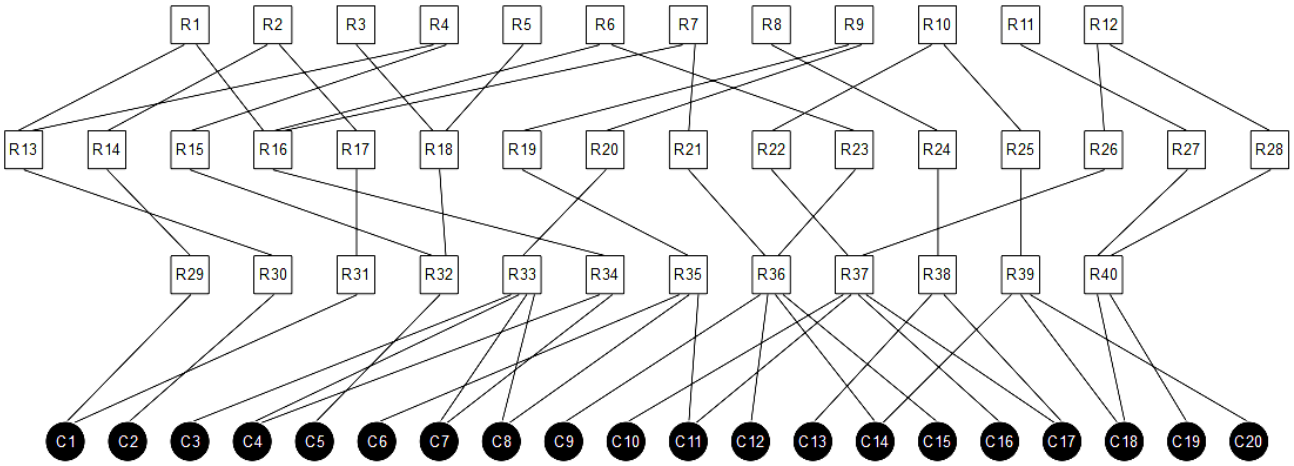

Fig 3: Instance NRP-B, with 20 clients and 40 interdependent requirements.

5.2 Resolution

The resolution by metaheuristic is carried out with frameworks EasyMeta [14] for the mono-objective problem. The framework is implemented in JAVA language, and implements several metaheuristics algorithms. Since metaheuristic algorithms are non-deterministic, one execution is not enough to analyze its performance. So, each metaheuristic was executed 100 times. The results shown indicate the average of 100 runs, and the variation of standard deviation of the values of the objective function.

The metaheuristics Simulated Annealing and Genetic Algorithms were chosen because of their widespread use in optimization problems. The configuration of genetic algorithms was cross-over rate of 0.9 and mutation rate of 5% (0.05). The maximum number of evaluations was 10,000, as well as in Simulated Annealing. These parameters were chosen by previous experimentation and tuning. A resolution by random approach was also performed.

The resolution in exact optimization was performed, as the problem tackled in this paper is of integer programming, with Branch-and-Bound method [15].

Resolution by humans subjects were also carried out in order to validate the better result for the problem are found by the automated technique. The solutions of the experts were collected on forms specifically design for the task. In total, 21 people solved both NRP-A. In instances NRP-B the number of participants reached 13 software engineering specialists [16]. For the other instances of the problems the resolution was not carried out by specialists, because of the high complexity of such instances. The analysis presented next was based on the average value and standard deviation of the objective function among the solutions of the participants.

6. RESULTS

6.1 Solutions

The results of the instances NRP-A and NRP-B are presented initially, because these are the ones that have answers of all techniques, including experts, which answered to the scenario of 70% of budget. The Table 2 shows the values of the objective function (to be maximized) for these results.

Table 2. Results for NRP-A and NRP-B with 70%.

Method	NRP-A 70%	NRP-B 70%
Exact Technique	27	96
Genetic Algorithm	26.45±0.500	95.41±0.190
Simulated Annealing	25.74±0.949	90.47±7.023
Human Experts	16.19±6.934	77.85±13.459
Random	15.03±5.950	45.74±11.819

The results from Exact Optimization and Metaheuristics are similar. Genetic Algorithm performed 2.03% worse than exact technique in NRP-A, and 0.61% in NRP-B. The Simulated Annealing metaheuristic was 4.67% and 5.76% worse in NRP-A and NRP-B, respectively.

However, the main information in Table 2 is the results achieved by experts. In the instance NRP-A, the average objective function value was 16.19, whilst the global optimum found by the exact optimization is 27. Thus, the average solution of specialists was 40.74% worse. In the instance NRP-B, the average of the experts was 18.90% worse when compared to the optimal solution found by exact optimization. The results indicate the validity of the use of automated optimization techniques to solve this problem and, as expected, the best results are obtained by exact optimization.

Table 3 below shows the results for the instance NRP-C for the three scenarios, 30%, 50% and 70% of budget. In such instance, as well as for NRP-D and NRP-E, the experiments were performed with the exact optimization and metaheuristics.

Table 3. Results for NRP-C, for 30%, 50% and 70%.

Method	NRP-C 30%	NRP-C 50%	NRP-C 70%
Exact Technique	96	146	206
Genetic Algorithm	80.45±9.34	124.25±9.60	199.07±4.37
Simulated Annealing	71.89±9.50	107.00±10.69	170.63±13.11

The results confirm that the exact optimization finds the best solutions to the problem. As for the behavior related to budget scenarios, the metaheuristics achieve worse results for scenarios with restricted available budget. For example, in the instance NRP-C, the average objective function values of genetic algorithms compared to the exact optimization are 3.36% lower in the case of budget 70%, 14.89% worse for the scenario of 50%, and 16.19% lower in the case of 30%. Therefore, the use of exact optimization is even more valid in practical contexts with more restricted budget. For Simulated Annealing, the results are: 17.17%, 26.71%, and 25.11% worse than exact optimization in scenarios of 70%, 50% and 30% of budget, respectively. These values are presented below in Figure 4 for visualization.

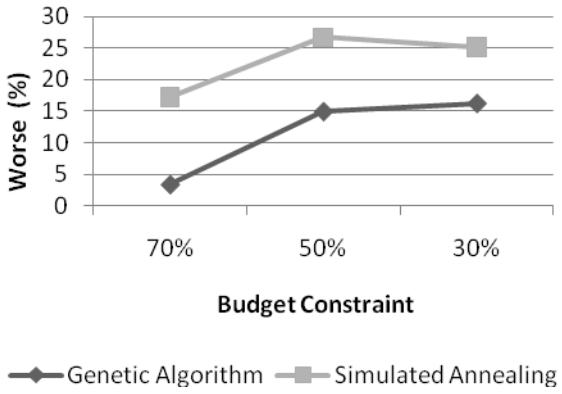

Fig 4: Results for metaheuristics against exact in NRP-C.

From the Figure 4 it can be perceived the evolution of worsening of both the metaheuristics with the greater budget constraint. Another aspect observed is that between 70% and 50% there is a major change in the variation of the techniques. Between 50% e 30%, however, a minor change occurs. This shows that the performance of the metaheuristics is strongly affected even by the reduction of 70% for 50%. This result shows the importance of exact optimization approach in the context. The values of the variations are: from 70% to 50%, Genetic Algorithm gets 343.15% worse and Simulated Annealing gets 55.56%. From 50% to 30% of budget, the values are -8.02% for Genetic Algorithm (the metaheuristic got better) and 6.37% for Simulated Annealing.

In order to show the overall performance of the techniques, we also present the results for the bigger instances, The instance NRP-D, for example, is twice the size of the previous NRP-C. Table 4 next presents the results for NRP-D.

Table 4. Results for NRP-D, for 30%, 50% and 70%.

Method	NRP-D 30%	NRP-D 50%	NRP-D 70%
Exact Technique	128	205	278
Genetic Algorithm	87.60±13.91	160.88±16.92	270.70±15.17
Simulated Annealing	79.02±12.36	133.96±17.33	211.70±15.99

As for NRP-C, the results for the instance NRP-D shows the better performance of exact techniques. Regarding the behavior to budget scenarios, the metaheuristics also achieve worse values compared to the exact optimization by the increase in the restriction of the budget. The average results of genetic algorithms are 2.62%, 21.52% and 31.56 worse in the limits of 70%, 50%, and 30%, respectively. The Simulated Annealing had the following results of worse variations than exact optimization: 23.84%, 34.65%, and 38.26 in scenarios of 70%, 50% and 30% of budget. So, in this instance it is also applicable the fact that the use of exact optimization is valid in practical contexts, because they may eventually present restricted budget. As done for NRP-C, the Figure 5 shows these values for better visualization.

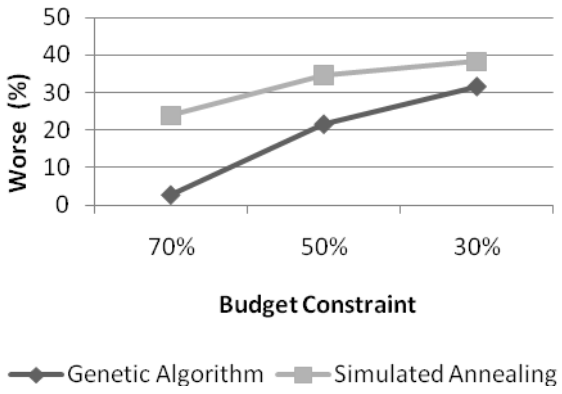

Fig 5: Results for metaheuristics against exact in NRP-D.

The Figure 5 indicates a growth of worsening more smooth than Figure 4. It indicates that for an instance with the twice size, the metaheuristics stated to show a regular level of worsening with more constrained scenarios. Despite that, the minor variation between 70% and 50% compared to that between 50% and 30% also happens in NRP-D. This result reinforces the importance of exact optimization in the problem. The values for variations for Genetic Algorithm are: from 70% to 50%, 721.37% worse and from 50% to 30%, it is 46.65% worse. Simulated Annealing gets 45.34% between 70% and 50% of budget constraint, and 10.41% for 50% and 30%. Another aspect is that in this case there was not a better result between 50% and 30%, as presented in the previous case.

Finally, the results for NRP-E are shown in Table 5 below.

Table 5. Results for NRP-E, for 30%, 50% and 70%.

Method	NRP-E 30%	NRP-E 50%	NRP-E 70%
Exact Technique	205	299	314
Genetic Algorithm	177.09±9.60	293.94±3.19	313.85±0.41
Simulated Annealing	147.29±14.81	265.75±13.47	314.00±0.00

The results for the NRP-E indicate the better performance of exact techniques in all scenarios. Despite the result for Simulated Annealing in the 70% constraint budget shows that the metaheuristic has achieved the best solution, the performance of exact optimization was better in all scenarios. The behavior regarding the budget scenarios is as follows: the Genetic Algorithm had in average results 0.04%, 1.69%, and 13.61% worse in the limits of 70%, 50%, and 30%, respectively. The metaheuristic Simulated Annealing was worse than exact optimization by 0.00%, 11.37%, and 28.15% in scenarios of 70%, 50% and 30% of budget. This result shows that, as for NRP-C and NRP-D, in NRP-E the metaheuristics got worse with the increase in the restriction of the budget. As for the previous instances, the Figure 6 shows the variation values.

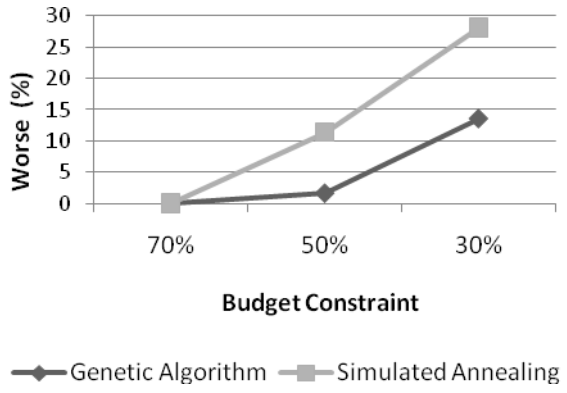

Fig 6: Results for metaheuristics against exact in NRP-E.

6.2 Execution Time

Beyond the analysis of accuracy of solutions, the runtime of executions of the techniques is also an aspect to be investigated.

The time results shown below are presented in milliseconds. This data were collect from the metaheuristics by the average and standard deviation of 100 executions. In the exact technique, the time is of the only execution of the method. In the results for human subjects, the data were collected by the time the experts needed to resolve the problem. Then, the values were taken in average and standard deviation. While the other automated techniques used milliseconds or seconds of execution, the humans took minutes to solve the instances. This is why the values presented below for human are much greater than the values for the others techniques.

Initially, the results of NRP-A and NRP-B are presented in Table 6. The results are from the constraint of 70% of budget.

Table 6. Runtime results for NRP-A and NRP-B with 70%.

Method	NRP-A 70%	NRP-B 70%
Exact Technique	520	1030
Genetic Algorithm	40.92±11.112	504.72±95.665
Simulated Annealing	23.01±7.476	292.62±55.548
Human Experts	1,731,428.57 ±2,587,005.57	3,084,000.00 ±2,542,943.10
Random	0.00±0.002	0.06±0.016

The results indicate the exact optimization technique has runtime greater than the average of metaheuristics. In NRP-A, the runtime of exact technique was 12.70 times of the Genetic Algorithms, and 22.59 times than the execution runtime of Simulated Annealing. However, the time of exact technique was about half second, and then it is a valid execution time for the context. Other aspect regarding this issue is that the time from exact technique was lower that the time required for the human experts. The results for B show similar behavior: execution runtime of exact approach is 2.04 times than Genetic Algorithm and 3.51 than Simulated Annealing.

Table 7 and Figure 7 show the results for execution runtime for NRP-C.

Table 7. Execution results for NRP-C (30%, 50% and 70%).

Method	NRP-C 30%	NRP-C 50%	NRP-C 70%
Exact Technique	5520	8410	1850
Genetic Algorithm	1718.56 ±259.97	1735.41 ±182.42	2162.30 ±287.98
Simulated Annealing	3264.88 ±668.00	2150.93 ±345.49	1384.98 ±186.31

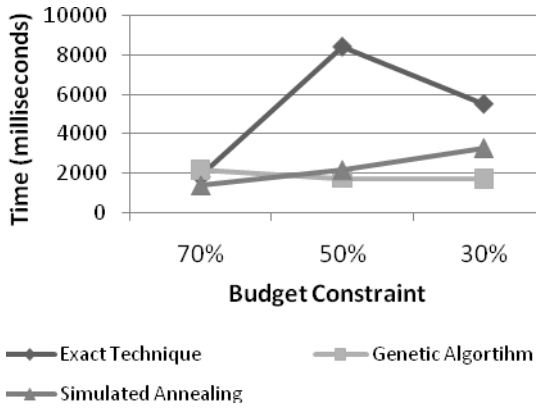

Fig 7: Runtime results in NRP-C.

Despite the greater execution time in most cases, the exact technique still has a reasonable time. For example, the greater value in NRP-C is 8,410 milliseconds, which is about 8.4 seconds. This value, even greater than the ones required by the metaheuristics, is of valid use in the context.

The results of execution runtime for NRP-D are shown in Table 8 and Figure 8 below.

Table 8. Execution results for NRP-D (30%, 50% and 70%).

Method	NRP-D 30%	NRP-D 50%	NRP-D 70%
Exact Technique	8520	1441	5070
Genetic Algorithm	36649.20 ±10792.79	42794.11 ±15531.51	37698.60 ±3790.49
Simulated Annealing	80568.10 ±26046.21	56605.51 ±21300.49	31820.10 ±5967.78

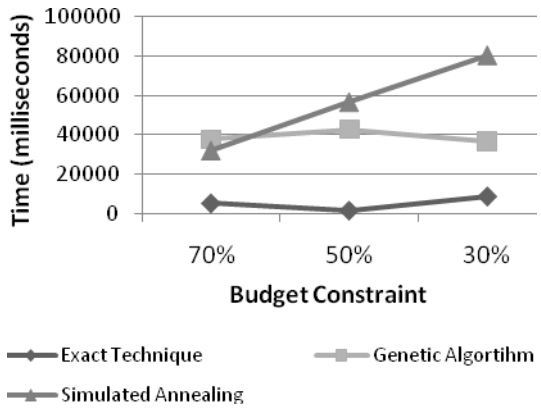

Fig 8: Runtime results in NRP-D.

In this case, the time of exact technique was lower than the metaheuristics. That happens mainly because for a bigger instance, the metaheuristics are set to have more evaluations and operations. Nonetheless, a result that can be taken from the graphic is that the Genetic Algorithm had similar times, while Simulated Annealing has presented an increasing runtime by the increase in budget constraint.

Finally, Table 9 and Figure 9 show the results for execution runtime for NRP-E.

Table 9. Execution results for NRP-E (30%, 50% and 70%).

Method	NRP-E 30%	NRP-E 50%	NRP-E 70%
Exact Technique	1850	1070	1380
Genetic Algorithm	6760.64 ±1653.32	6130.74 ±1165.81	6191.47 ±650.08
Simulated Annealing	7132.54 ±1898.25	3446.03 ±670.57	2841.17 ±283.82

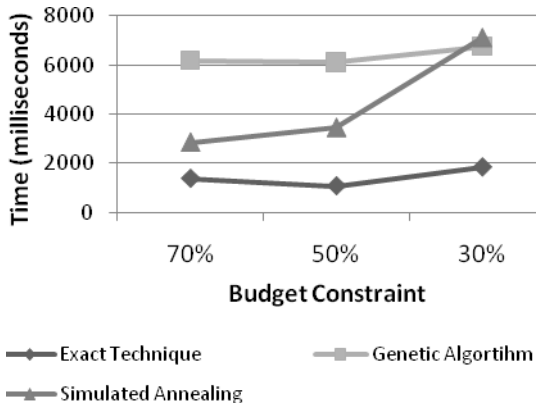

Fig 9: Runtime results in NRP-E.

A result that can be taken from the Figure 9 is that Simulated Annealing strongly increase its time by the 30% budget constraint.

7. CONCLUSION

Given the importance of software systems in today's society, it is important that their development is done in the best possible way. During such a development, some complex problems can occur and to solve them by those software experts involved may be a highly difficult task. In the phase of requirements engineering, for instance, a complex problem occurs when dealing with the selection of which requirements should be implemented to the next release, based on values of importance and in budget constraints.

In literature, this problem has been tackled by metaheuristics, and in this work we propose the resolution by exact techniques, which have the advantage of finding the best solutions to the problem. From tests carried out in several instances, we can indicate the validity of such an approach. Besides, as expected, the exact optimization results were better than the metaheuristics, the execution time has shown to be reasonable.

In this case, it was observed that the solutions of the metaheuristic genetic algorithms, which performed better compared to Simulated Annealing, were up 31.56% worse compared to those found by the exact method. Moreover, it was noticed that for bigger problems, more complex, the overall performance of metaheuristics deteriorated. An additional fact taken from metaheuristics is that the average performance of these techniques gets worse with a more restricted limit of budget constraint of the problem.

Regarding execution time, the approach based on exact techniques presented, indeed, in general, more time to resolve the instances of the problems. However, for example, the longer runtime stated in the mono-objective problem was 8.52 seconds, in the instance NRP-D, which shows that, despite being a higher value when compared to those of metaheuristics, it constitutes an acceptable execution time.

This study also carried out comparisons with solutions from experts involved in the area. From the results, we verify the necessity of applying the automatic techniques and based on mathematical optimization in solving the problems, because in

general the results found by experts proved worse. In the case of the mono-objective selection requirements, we found that on average the solutions given by experts were 40% and 18.9% worse than the optimal solution in NRP-A and NRP-B, respectively.

As future work, we indicate performing experiments in larger instances, as well as tests with different data in order to demonstrate the validity of the approach in different scenarios. Other research is to deal with testing the approach with real data from software projects.

8. REFERENCES

[1] T. Dyba, "An empirical investigation of the key factors for success in software process improvement", IEEE Transactions on Software Engineering, May 2005, pp. 410-424.

[2] Fuggetta, A. 2000. Software process: a roadmap, The Future of Software Engineering, A. Finkelstein (ed).

[3] A. Bagnall, V. Rayward-Smith, L. Whittley, "The next release problem", Information and Software Technology, 2001, pp. 883–890.

[4] Yoo, S. and Harman, M. 2007. Pareto Efficient Multi-Objective Test Case Selection. In Proceedings of the International Symposium on Software Testing and Analysis, pp. 140-150.

[5] Zhang, Y., Harman, M. and Mansouri, A. 2007. The multi-objective next release problem. In Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO '07). ACM, pp. 1129-1137.

[6] Harman, M. 2006. Search Based Software Engineering, In Workshop on Computational Science in Software Engineering.

[7] J. Clarke, et al. "Reformulating software engineering as a search problem", IEE Proceedings Software, Vol. 150, No. 3, June 2003, pp. 161-175.

[8] W. Miller and D. Spooner, "Automatic Generation of Floating-Point Test Data", IEEE Transactions on Software Engineering, Vol. 2(3), pp. 223-226, 1976.

[9] M. Harman, and B.F. Jones, "Search-based software engineering", Information and Software Technology, 2001, pp. 833-839.

[10] Glover, F. 1986. Future paths for integer programming and links to artificial intelligence, Computer Operational Research 13, pp. 533-549.

[11] S. Kirkpatrick, D. C. Gellat, M. P. Vecchi, "Optimizations by simulated annealing", Science v. 220, pp. 671-680, 1983.

[12] Holland, J., 1975. Adaptation in Natural and Artificial Systems.

[13] Dantzig, G. B. Inductive proof of the simplex method. IBM J. Res. Development, 505-506, 1960.

[14] Carmo, R. A. F, Campos. G. A. L., Souza, J. T. Easymeta: a framework of metaheuristics for mono-Objective optimization problems. In Proceedings of the XL Simpósio Brasileiro de Pesquisa Operacional (SBPO´2008), 2008.

[15] Land, A. H., Doig, A. G.; An automatic method of solving discrete programming problems. Econometrica 28(3): 497-520, July 1960.

[16] Souza, J., Maia, C., Freitas, F. and Coutinho D. 2010. The Human Competitiveness of Search Based Software Engineering. In Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10), pp. 143-152.