Hill Climbing and Simulated Annealing in Large Scale Next Release Problem

Goran Mauša [#1], Tihana Galinac Grbac [#2], Bojana Dalbelo Bašić [*3], Mario-Osvin Pavčević [*4]

[#] *Faculty of Engineering, University of Rijeka*
Vukovarska 58, 51000 Rijeka, Croatia
[1] goran.mausa@riteh.hr
[2] tihana.galinac@riteh.hr

[*] *Faculty of Electrical Engineering and Computing, University of Zagreb*
Unska 3, 10000 Zagreb, Croatia
[3] bojana.dalbelo@fer.hr
[4] mario.pavcevic@fer.hr

***Abstract*—Next release problem is a software engineering problem, lately often solved using heuristic algorithms. It deals with selecting a subset of requirements that should appear in next release of a software product. The problem lies in satisfying various parts interested in project development with acceptable costs. This paper compares two rather simple, but often used and efficient heuristic algorithms: Hill Climbing and Simulated Annealing. The aim of this paper was to compare the performance of these algorithms and their modifications on a large scale problem. We investigated the differences between four variations of Hill Climbing and two variations of Simulated Annealing, while Random Search was used to verify the benefit of using a heuristic algorithm. The evaluation was performed in terms of finding the best solution for a given budget and in calculating the proportion of non-dominated solutions that form the joint Pareto-optimal front. Our research was done on publicly available realistic datasets that were obtained mining the bug repositories. The results indicate Simulated Annealing as the more successful algorithm but point out that Simulated Annealing together with Hill Climbing provides a more thorough insight into the problem search space.**

***Index Terms*—next release problem, Hill Climbing, Simulated Annealing, large scale problem, realistic dataset**

I. Introduction

In this paper we address the next release problem (NRP), a problem present in requirements engineering. This task of choosing a subset of possible requirements often meets more than a couple of contradictory demands. That is why this problem falls into the domain of multi-objective problems, often solved using search based software engineering algorithms [1]. Satisfying as many of the contradictory demands is not the only constraint under which this problem is dealt with. There is also the problem of budget, i.e. limited financial resources available in the project [2]. The mere concept of trying to satisfy the highest number of stakeholders or customers while spending the lowest amount of resources in the process is a contradictory demand which leads to a necessary trade-off.

The NRP is very similar to the Knapsack problem and often has a very large search space due to a greater number of possible requirement configurations [3]. That is why exhaustive search cannot solve the problem efficiently and metaheuristics are used instead. Using search based software engineering in such a multi-objective problem does not always give one optimal solution to the problem. It usually gives several incomparable or equally good solutions that should provide an insight into the search space of candidate solutions. Therefore, the final decision is left to an expert [4].

There are several research questions that drive our study. Examining the related work we noticed a rare use of Hill Climbing algorithm when dealing with NRP. Although Simulated Annealing algorithm usually outperforms other local search algorithms, the comparison was never performed with realistic data. Research performed by Xuan et. al [5] made such a dataset publicly available. This research gap motivates our first research question:
RQ1: Does Simulated Annealing algorithm outperform Hill Climbing algorithm in realistic large scale NRP?

We also noticed that the comparison of Hill Climbing algorithm and Simulated Annealing algorithm was never performed in terms of forming the Pareto-optimal front of non-dominated solutions, which provide a more thorough insight into the problem search space and are more informative for the decision maker. That brings us to the second research question:
RQ2: How does Hill Climbing perform in forming the Pareto-optimal front of non-dominated solutions in realistic large scale NRP?

The realistic datasets we used in our research are instances of a large scale problem with number of stakeholders in range between 300 and 800 and number of requirements between 2000 and 4500. Many other studies used smaller datasets, like the one proposed by Greer and Ruhe and adopted by other studies which contain only 20 requirements and 5 stakeholders. We believe the huge search space in our research makes the algorithms with thorough search of neighbourhood solutions like Steepest Ascent Hill Climbing unable to adopt more than a couple of solutions and therefore makes them inappropriate for dealing with such a problem. We also wanted to investigate how the order of neighbourhood search influences the algorithms' results. One way of neighbourhood was purely randomized, while the other one followed the order of

978-1-4673-2232-4/13/$31.00 ©2013 IEEE

requirements in the dataset. Finally, we present the last two research questions:
RQ3: Does large scale problem make Steepest Ascent Hill Climbing algorithm incapable of efficiently solving the NRP?
RQ4: Does order of neighbourhood search influence the local search heuristic algorithm in finding the optimal solution in large scale NRP?

This paper consists of six sections. Next section explains the motivation for our choice of algorithms and evaluation criteria examining the related work. Following section contains some basic information about heuristic algorithms in general and presents the Hill Climbing and the Simulated Annealing algorithms. Section IV describes our case study in details: how the NRP is structured, which are the datasets we had used to compare the two algorithms and finally which are the results we have obtained. The answers to our 4 research questions, along with future work are concluded in the final section.

II. Background

The fitness function and search space are the only two requirements for usage of a heuristic algorithm. The mere simplicity of requirements and the fact that fitness function may be defined in any possible way make the search based software engineering applicable to a very large number of problem domains. The choice of using heuristic algorithm instead of exhaustive search may be defined by the problem itself. Many problems have too large search space for the exhaustive search to finish in reasonable time and for some of them a "near optimal" solution is sufficient. The heuristic algorithms can also rather quickly give an insight to problems where the configuration of the search space and its candidate solutions is unknown. All of these advantages make heuristic algorithms used in all stages of software product life cycle and form the area also known as search based software engineering (SBSE). The most often used algorithms are genetic algorithms, genetic programming, Hill Climbing and Simulated Annealing [7], [18]. The latter two are used in this paper.

Solving the NRP in software engineering using heuristic algorithms was first proposed by Bagnall et al. [6] in 2001. Their study compared three instances of Hill Climbing (Steepest Ascent, First Found and Sampling), two instances of Simulated Annealing and a greedy algorithm upon 5 randomly generated datasets. The evaluation criterion was best solution found by an algorithm for a determined budget constraint. The results showed the Simulated Annealing to be the most successful algorithm.

Hill Climbing was rarely used in dealing with the NRP. Besides Bagnall et. al [6], there were only a few other researches performed using this algorithm. Lu et. al [7] used two instances of Hill Climbing algorithm and compared them in terms of finding one best solution and calculating time consumed for that task. However, their focus was the fitness landscape analysis and the improvement it could bring to the heuristic algorithm. Jiang et. al [8] proposed greedy climbing search, a Hill Climbing algorithm based on greedy strategy and stated that Simulated Annealing proved to be efficient only for the small scale NRP. Their study compared the approximate backbone based multilevel algorithm, greedy Hill Climbing and Simulated Annealing algorithms in terms of best profit for determined budget and time consumption. Xuan et. al [5] are the last ones to have used the Hill Climbing algorithm, but they used it only for the fitness landscape analysis. There are other heuristic algorithm often used in NRP, like NSGA-II [1], [3], [4], [9]–[14], Ant Colony [2], MOCell [3], [11], Greedy algorithm [4], [6], Two-archive algorithm [9], [10] and Simulated Annealing [2], [6], [11], [13]. However, we find only the Simulated Annealing algorithm similar enough to be compared with the Hill Climbing algorithm and it showed to be one of the best algorithms in other studies.

Due to lack of publicly available real world data, majority of studies that deals with solving the NRP using heuristic algorithms used synthetic, randomly generated instances [1]–[3], [6]–[12], [14], [15]. Even the Xuan et. al [5] used synthetic datasets, which were named as classical. One of the most popular such datasets is the one proposed by Greer and Ruhe [15] in 2004. Even though it was a randomly generated dataset, they presented all the values that constitute it so other researchers like Sagrado et. al [2], Zhang et. al [9] and Finkelstein et. al [10] used the same values. However, they could not compare the results because the information regarding the cost of each requirement was missing. The missing values were generated randomly, with the range [10, 1100], following Gaussian distribution but the dataset was not identical. The only non-synthetic dataset was provided by a large telecom company [4], [9], [13] and most likely due to confidentiality agreement it was never made publicly available.

The research done by Bagnall et al. [6] influenced the research of Xuan et. al [5] and our own as well. While Bagnall et al. used randomly generated datasets, we used the realistic ones from [16]. To our knowledge, this is the first time the Hill Climbing algorithm is evaluated in term of forming the Pareto-optimal front, which could provide a better insight to the problem and possible solutions that having just the highest value of profit or satisfaction for a limited budget. The Pareto-optimal front is a subset of non-dominated solutions. A non-dominated solution is superior to other examined solutions in terms of objectives we are optimizing. In our case, the objectives are to minimize the decision cost *DC* and to maximize the profit. For each solution present in Pareto-optimal front we can say that no other examined solution can improve one objective without worsening the other objective. We also did not encounter any study that compared the Hill Climbing and Simulated Annealing algorithms using realistic data.

"Ordered" and "Random" procedures of the neighbourhood search are presented both in Hill Climbing and Simulated Annealing. We had not encountered a case study that deals with this question so far. We believe the differences they provide to the algorithm results would be insignificant for the small scale problems. With reasonably high number of maximal iterations c_{max} both procedures would cover most of the solutions. However, with the same value of c_{max} they

978-1-4673-2232-4/13/$31.00 ©2013 IEEE

would cover a lot less than total number of possible solutions for a large scale problem. With limited span of solutions they could cover, the difference in the neighbourhood search procedure could lead to different subsets of solutions they would examine. That is why we focused our research to this topic in our RQ4 as well.

III. The Next Release Problem

Next release problem can be represented with following input parameters:

- A set of *m* stakeholders whose demands are to be considered in the requirements engineering phase

$$S = \{s_1 \quad s_2 \quad \dots \quad s_m\} \tag{1}$$

- A set of *n* requirements that are possible for the next release of a software product

$$R = \{r_1 \quad r_2 \quad \dots \quad r_n\} \tag{2}$$

- A set of *n* costs that denote the resources needed for the proper implementation of each requirement

$$C = \{c_1 \quad c_2 \quad \dots \quad c_n\} \tag{3}$$

- A set of *m* x *n* values that quantify the satisfaction of a stakeholder by choosing a specific requirement or binary values that represent which of *n* requirements are desired by each of the *m* stakeholders

$$V = \begin{Bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,n} \\ v_{2,1} & v_{2,2} & \dots & v_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m,1} & v_{m,2} & \dots & v_{m,n} \end{Bmatrix} \tag{4}$$

- A set of *m* values that represent the potential profit from each stakeholder or their importance weight

$$P = \{p_1 \quad p_2 \quad \dots \quad p_m\} \tag{5}$$

Besides these four parameters that characterize the multi-objective problem we are dealing with, there are some additional internal parameters:

- The budget available at the start of project, often represented as percentage β of total costs needed to implement all of the requirements

$$B = \beta \sum_{i=1}^{n} c_i \tag{6}$$

- A decision vector of n binary values that represent a *Solution*, with value of 1 for the requirements that ought to be implemented and with the value of 0 for those that ought to be excluded

$$X = \{x_1 \quad x_2 \quad \dots \quad x_n\} \tag{7}$$

- The decision cost as the amount of resources needed to implement the decision under consideration

$$DC = \sum_{i=1}^{n} c_i x_i \tag{8}$$

- The level of profit the decision under consideration would bring from all of the *m* stakeholders. The profit from each stakeholder is obtained only if all of his demanding requirements are included in the decision vector.

From these internal parameters, the objective of the NRP can be summed up to make a final decision $X_{best} = \{x_1 \quad x_2 \quad \dots \quad x_n\}$, that would produce a level of satisfaction or profit as high as possible while retaining the costs below the budget constraint DC<B.

IV. Heuristic Algorithms

Heuristic algorithms are a group of optimization algorithms and techniques that use a certain degree of randomness. They are used for large scale problems for which we have no insight what the optimal solutions looks like and where exhaustive search proved to be too slow. The application of randomness speeds up the search process and it can be described as proceeding by trial and error. There are several general terms common to any heuristic algorithm:

The Search Space is the whole range of parameters' values that presents solutions to the problem. In our case, the search space is any possible combination of requirements proposed to be included in the next release of a software product, i.e. any solution *X*.

The Fitness Function is a quantitative representation of solution's quality. In our case, the fitness function is the level of profit a solution would provide.

The Feasible Solution is a solution *X* that implies a decision cost *DC* below the budget *B* constraint for the NRP.

The Optimal Solution is a *feasible solution* X_{best} that provides highest level of profit for encountered decision cost *DC*.

The Candidate Solution is a *feasible solution X* that is under consideration for becoming the *optimal solution* X_{best}.

The Neighbourhood consists of all solutions that are close to *the candidate solution*. In our case, the neighbourhood of *the candidate solution X* are all the solutions that have only one bit flipped.

Ordered Search Procedure means the neighbourhood is to be searched in repeating incrementing order from the first requirement to the last, following the same order as is given by the dataset.

Random Search Procedure performs the neighbourhood search in randomized order.

Random Restart is used if the algorithm stops before it is supposed to. Then it restarts from another randomly selected candidate solution.

The first Candidate Solution is usually a randomly chosen *feasible solution* from *the search space* at the initialization phase of a heuristic algorithm. The second phase is the quality check of *candidate solution*. It implies the calculation of its *fitness function*. The fitness function is defined by the problem we are dealing with and it quantifies the quality of a candidate solution [17]. The following phases of a heuristic algorithm are included in an iterative procedure. The modification phase examines the solutions in *the neighbourhood* of *the candidate*

solution searching for *the optimal solution*. After the quality check of examined solutions in *the neighbourhood* we get to the decision phase. It decides which *candidate solution* proceeds to the following iteration, taking *fitness function* into account. The search stops when it reaches a solution that is good enough, a non-dominated solution or if it exceeds the predefined maximal number of iterations. In our case, the stopping criterion of all algorithms is the predefined maximal number of iterations c_{max}. The algorithm randomly restarts until the value c_{max} is achieved.

There are two major differentiation factors that classify the heuristic algorithms in one of the following groups:

- **Single State vs. Population Based Algorithms** - depending on the number of *candidate solutions* examined simultaneously,
- **Local vs. Global Search** - depending whether the algorithm finds local or global optima.

While *single state algorithms* analyze only one *candidate solution* at the time, the *population based algorithms* analyze a whole span of solutions at each iteration. The *local search* algorithms have the disadvantage of becoming trapped in a local maximum of *the search space*. The *global search* algorithms, on the other hand, overcome that disadvantage but at the cost of higher computation time.

A. Hill Climbing

Hill Climbing is a local search, single state heuristic algorithm. The trade-off between efficiency and effectiveness can be noticed in such an algorithm. Due to the simplicity of the algorithm it is very fast, but it has the potential disadvantage of becoming trapped in a locally optimal solution [19]. The algorithm is presented in Table I. It starts with random choice of a feasible solution. It begins the iterative procedure searching for another feasible solution in the neighbourhood. Finally it selects the solution with highest fitness function as its next candidate solution.

TABLE I
HILL CLIMBING ALGORITHM

Hill climbing (c_{max})

Input: neighbourhood(), Profit()
Select a feasible solution X $\in \chi$
c $\leftarrow$ 0
$X_{best} \leftarrow$ X
searching $\leftarrow$ **true**
while c $< c_{max}$ & *searching*
do
    Y $\leftarrow$ neighbourhood(X)
    if Y $\neq$ Fail
    then
        X $\leftarrow$ Y
        if Profit(X) $>$ Profit(X_{best})
        then $X_{best} \leftarrow$ X
    else
        searching $\leftarrow$ **false**
    c $\leftarrow$ c + 1
return (X_{best})

Four instances of Hill Climbing algorithm that we used in our research are different in the way they search for a better solution in the neighbourhood:

1) **Steepest Ascent**
The Steepest Ascent looks for the best solution in the whole neighbourhood before moving to the following solution.

2) **First Found - Ordered**
The First Found neighbourhood search is a much faster procedure than the Steepest Ascent. As it finds a solution that is superior to the current one, it immediately adopts it as next candidate solution. The supplement Örderedïs our label for one type of neighbourhood we want to compare in RQ4.

3) **First Found - Random**
This is another First Found neighbourhood search procedure, with a difference from the previous one that the order of neighbourhood search is not influenced by dataset, but is entirely random. ”Random” is the second type of neighbourhood search procedure compared in RQ4.

4) **Sampling**
The Sampling procedure does not search the whole neighbourhood, but instead examines only a randomly selected subset of neighbouring solutions and takes the best one among them as its next candidate solution. We encountered the algorithm in [6] and noticed the subset size remained unexplained. Our intention was to make the subset size linked to the number of requirements present in a dataset. Since the number of requirements varies between 2000 and 4500, we looked for a way to minimize this large range. Drawing the motivation from the random forest algorithm we determined the subset size to be equal to square value of number of requirements, i.e. number of neighbouring solutions. This approach provided us with equal order of magnitude of subset size for each dataset as the one given in [6].

B. Simulated Annealing

The Simulated Annealing algorithm is another local, single state heuristic algorithm. Unlike Hill Climbing, the Simulated Annealing has the possibility of choosing a solution inferior to the previous one. The probability of choosing the inferior solution is determined through additional parameter *T*. The inferior solution will be chosen if $r < e^{(Profit(Y)-Profit(X))/T}$, where *r* is a randomly chosen number in range [0,1] and *(Profit(Y)-Profit(X))* is the difference between profit of the inferior and the current candidate solution. With each following iteration, the parameter *T* is reduced by a factor α and the chance of choosing an inferior solution is decreased. This enhancement sometimes enables the algorithm to escape from the locally optimal solutions in the beginning and reach a more optimal solution at the end of the search procedure. When dealing with NRP, the value of parameter T_0 is usually set to 100 and the value of parameter α is set close to 1 [6], [17]. For higher values of these two parameters, it will take longer time before the algorithm starts to reject the inferior solutions completely. The algorithm is presented in Table II.

There are two instances of Simulated Annealing we used in our research. Like in the case of Hill Climbing, they are

TABLE II
SIMULATED ANNEALING ALGORITHM

```
Simulated annealing (c_max, T_0, α)

Input: Neighbourhood(), Random(), Profit()
c ← 0
T ← T_0
Select a feasible solution X ∈ χ
X_best ← X
while c < c_max
do
    Y ← neighbourhood(X)
    if Y ≠ Fail
        then if Profit(Y) > Profit(X)
            then X ← Y
                if Profit(X) > Profit(X_best)
                    then X_best ← X
            else r ← Random(0,1)
                if r < e^((Profit(Y)−Profit(X))/T
                    then X ← Y
    c ← c + 1
    T ← α T
return (X_best)
```

different in the neighbourhood search procedure and both are in focus of RQ4:

1) **Simulated Annealing - Ordered**
2) **Simulated Annealing - Random**

C. Random Search

The Random Search algorithm was used merely as a sanity check to verify the benefit and the need of using a heuristic algorithm. In contrast to previous heuristic algorithms, this one is neither a local nor a global algorithm. It is the most basic search algorithm, analogous to pure guessing. In each iteration it looks for one feasible solution and calculates its profit and decision cost *DC*. Like other algorithms, it is limited with maximal number of iterations c_{max}.

V. CASE STUDY

A. Dataset

The datasets we used in our research are realistic datasets, mined from bug report repositories. Bugs are linked to requirements, user's comments to stakeholders, severity of bugs to requirements' cost and only the profit of each stakeholder was generated randomly. Table III gives the basic information of the datasets, while further details can be found in [5].

B. Evaluation

There are several ways of evaluating the results when dealing with NRP. The first one is the simplest and most often used. It aims to find one optimal solution for each budget constraint. In our case we are looking for highest profit for each budget limit. This kind of evaluation provides the decision maker with one solution that is pronounced as best one. The downside of this kind of evaluation is that it does not provide an insight into the search space. We are unaware of the profit versus cost trade-off between different budget constraints unless we use many budget constraint values. If we decide to use many budget constraint values, we prolong the execution time and make the whole process inefficient. This kind of evaluation is used in [2], [5]–[8], [13], [18] and it should not be neglected so we included it in our research.

The second evaluation procedure is an interesting procedure presented by Zhang et al. [10]. They evaluate the results plotting the Pareto-optimal front of non-dominated solutions for satisfaction of each stakeholder with limited budget. In that way they are able to examine the tensions between stakeholders and one of their conclusions was there were no tensions. The datasets we used contain a couple of hundreds of stakeholders and this kind of evaluation is out of the question due to practical reasons.

The third way of evaluating the results is not affected by high number of stakeholders or requirements and provides a more informative insight into the search space of the problem than the previous two. While the second evaluation procedure calculates the results for each stakeholder separately, the third one does it for overall profit/satisfaction brought by all the stakeholders together. That means the Pareto-optimal front is a subset of highest overall profit for corresponding value of cost, regardless of budget, and we obtain one Pareto-optimal front for each heuristic algorithm. In order to compare the results produced by all the algorithms, we formed the joint Pareto-optimal front. The joint Pareto-optimal front shows the highest profit for each cost value among all the algorithms. Then we calculated the percentage of solutions that are brought by each algorithm into the joint Pareto-optimal front. This kind of evaluation is present in [1], [3], [4], [9]–[12], [14] and we included this kind of evaluation in our research as well.

C. Experimental Design

Our research was divided into two phases. The first phase evaluated the results as described in the first evaluation procedure and the second phase evaluated the results as described in the third evaluation procedure from previous subsection.

The first research phase applied the Hill Climbing (Steepest Ascent, First Found - Ordered, First Found - Random and Sampling), Simulated Annealing (Ordered and Random) and Random Search algorithms to find the highest profit for limited budget. The algorithms were run with random restart until achieving 10,000 fitness evaluations, i.e. $c_{max} = 10,000$. The budget constraint spanned from β=0.3 to β=0.7 with step of 0.2. the Simulated Annealing's parameters are set to T_0=100 and α=0.9. The results obtained in this manner are presented in Table IV. The name of the dataset is given in the first column. The β value and the corresponding value of available budget *B* is given in the second column. The following columns are for all instances of each algorithm we used in our research and every column has one subcolumn that depicts the highest value of profit found by that algorithm and its decision cost *DC*. Values marked as **bold** are the best results for a given dataset and budget constraint, i.e. for each row.

In the second phase of our research we applied the Hill Climbing (First Found - Ordered and First Found - Random),

TABLE III
REALISTIC NRP DATASETS

Dataset	Nrp-e1	Nrp-e2	Nrp-e3	Nrp-e4	Nrp-g1	Nrp-g2	Nrp-g3	Nrp-g4	Nrp-m1	Nrp-m2	Nrp-m3	Nrp-m4
Num of stakeholders	536	491	456	399	445	315	423	294	768	617	765	568
Profit of stakeholders	[15,43]	[13,46]	[15,45]	[15,45]	[16,46]	[15,43]	[16,44]	[15,45]	[15,44]	[16,46]	[15,45]	[16,44]
Num of requirements	3502	4254	2844	3186	2690	2650	2512	2246	4060	4368	3566	3566
Cost of requirements	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]	[1,7]

Simulated Annealing (Ordered and Random) and Random Search algorithms to find the joint Pareto-optimal front. The algorithms were run with random restart with $c_{max} = 5,000$. Although the Pareto-optimal front is created regardless of budget constraint, we still need the algorithms to look for optimal solutions around certain values of cost. Otherwise they would simply include requirements until achieving maximum cost and neglect the search for near optimal solutions. Therefore, we included more budget constraints and so it spanned from β=0.3 to β=0.7 with step of 0.1. The Simulated Annealing algorithm had the same parameters as before: T_0=100 and α=0.9. Due to limited space and the fact we examine 12 different datasets, we omitted the graphical representations of Pareto-optimal fronts. Instead, we present the results in terms of percentage of solutions obtained from each algorithm that forms the joint Pareto-optimal front in Table V. The first column of the table contains the name of the dataset. The following columns are for all instances of each algorithm we used in this phase of our research. Each of them has two subcolumns that depict the percentage of solutions that are brought by each algorithm into the joint Pareto-optimal front in the ”Point” and in the ”Spread” range. The ”Point” range signifies the joint Pareto-optimal front has one value of highest overall profit for each value of cost examined by any algorithm. Downside to this procedure is that some algorithms could have found the highest profit just because they were the only ones that examined a particular cost value. The ”Spread” range tried to cope with that issue. The joint Pareto-optimal front in ”Spread” range has one value of highest overall profit for a range of 100 cost values.

D. Results

From the results present in Table IV, we conclude that Steepest Ascent and Sampling Hill Climbing algorithms are considerably worse than others. They are outperformed even by Random Search in all cases. Here, we must point out again that being outperformed by the Random Search algorithm means the algorithm is worse than pure guessing. This is the reason for excluding these two algorithms from the second phase of our research. The Steepest Ascent algorithm evaluates all the solutions present in the neighbourhood of current candidate solution and that is equal to the number of requirements. Since the number of requirements in our datasets is between 2000 and 4500 and $c_{max} = 10,000$, it will manage to evaluate only a couple of candidate solutions. Due to this reason we anticipated such results for the Steepest Ascent algorithm. Sampling Hill Climbing, on the other hand, had a considerable improvement on that major disadvantage for large scale problems but its results were equally poor. Even though Simulated Annealing's results are superior to other algorithms, the First Found Hill Climbing algorithm managed to find 4 out of 36 best solutions. The Random Search algorithm had never found a solution with highest profit for given budget and that proved the usage of heuristic algorithms was appropriate.

From the results present in Table V, it is interesting to notice that Random Search forms a rather high percentage of joint Pareto-optimal front in Point range. However, all of these points are inferior in their surrounding, which is evident in 0% for the Spread range. This fact convinced us that Spread range is more informative regarding the algorithms' quality of results. The results also show an unexpectedly high difference in results of the two Simulated Annealing instances. The Simulated Annealing - Random algorithm gave 0% of solutions to the joint Pareto-optimal front in Spread range, performing worse than both instances of Hill Climbing algorithm.

The results given both in Table IV and Table V enabled us to answer the four research question stated in section I of this paper:

E. Answer to RQ1

Our results indicate Simulated Annealing to be the superior algorithm to Hill Climbing on a large scale realistic NRP. Hill Climbing outperforms the Simulated Annealing algorithm only in few cases when searching for the highest profit for a given budget. However, observing the results of forming the joint Pareto-optimal front, we conclude the solutions proposed by Hill Climbing give a broader insight into the search space of large scale NRP and therefore should not be neglected.

F. Answer to RQ2

The results of Hill Climbing algorithm in forming the joint Pareto-optimal front of non-dominated solutions in realistic large scale NRP is inferior to Simulated Annealing. However, when looking at the Spread range we observe a respectable increase of percentage with respect to Point range. This increase in percentage signifies the solutions found by Hill Climbing algorithm are of greater quality than many solutions given by Random Search and even by Simulated Annealing with Random neighbourhood search procedure when observed in its cost related surrounding.

G. Answer to RQ3

Based on the results of this study, we can conclude that Steepest Ascent Hill Climbing is not an appropriate algorithm

TABLE IV
RESULTS: HIGHEST PROFIT AND CORRESPONDING COST FOR LIMITED BUDGET

			Hill climbing								Simulated Annealing				Random	
	Budget		Steepest Ascent		First Found Ord		First found Rnd		Sampling		Ordered		Random			
Data	Ratio	Value	Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost	Profit	Cost
nrp-e1	0.3	3945	0	0	243	3809	272	3756	82	3640	**894**	3937	730	3944	190	3872
	0.5	6575	310	6524	605	6574	1582	6567	392	6533	**2941**	6574	2308	6571	706	6531
	0.7	9205	1594	8766	3286	9204	3338	9204	1559	9000	4486	7530	**4782**	7928	2427	9132
nrp-e2	0.3	4778	0	0	95	4618	92	4604	26	4439	**224**	4777	191	4775	105	4450
	0.5	7964	156	7932	519	7656	514	7963	166	7957	1054	7956	**1075**	7948	452	7939
	0.7	11150	932	10717	**3235**	11147	1999	11149	724	10845	2272	9950	2144	8586	1265	10759
nrp-e3	0.3	3120	0	0	349	2971	303	2996	61	2992	**645**	3119	468	3114	175	3046
	0.5	5200	234	5163	1221	5199	719	5199	451	5128	**2754**	5197	2314	5177	700	5149
	0.7	7279	1421	7056	3730	7279	3232	7279	1652	7200	4317	6719	**4404**	6480	2021	7195
nrp-e4	0.3	3510	0	0	108	3349	35	3342	34	3356	**284**	3509	116	3490	115	3415
	0.5	5850	156	5746	621	5765	580	5849	213	5738	1172	5838	**1600**	5836	347	5780
	0.7	8189	594	7841	1835	8189	**2680**	8189	660	7962	2028	7848	2646	6805	1213	8077
nrp-g1	0.3	3983	19	3633	304	3935	310	3895	105	3892	**859**	3979	700	3979	219	3981
	0.5	6639	322	6507	1318	6638	652	6638	574	6610	**2924**	6610	2633	6625	790	6626
	0.7	9294	2074	9230	2199	9293	3122	9293	2005	9064	4337	8659	**4748**	8492	2363	9106
nrp-g2	0.3	3788	31	3527	57	3517	37	3656	53	3766	**293**	3786	108	3775	105	3644
	0.5	6313	61	6267	397	6312	646	6310	206	6145	1249	6300	**1482**	6207	391	6270
	0.7	8838	466	8685	1525	8838	**2366**	8698	694	8606	1628	8588	2333	7360	1345	8585
nrp-g3	0.3	3677	28	3520	455	3526	463	3523	102	3500	**938**	3673	508	3677	217	3518
	0.5	6129	245	6058	523	6128	835	6128	404	5892	**3259**	6128	3050	6127	767	6125
	0.7	8581	1332	8147	3131	8580	3589	8580	1608	8212	**4421**	8315	3187	7085	2314	8567
nrp-g4	0.3	3210	0	0	102	3182	33	2924	65	3010	**271**	3187	202	3208	99	3082
	0.5	5350	118	5315	497	5322	751	5344	219	5272	1316	5316	**1511**	5307	364	5307
	0.7	7490	917	7217	1433	7489	2191	7235	809	7168	1809	7361	**2849**	6647	1134	7411
nrp-m1	0.3	4722	33	4501	433	4721	606	4664	100	4564	**1064**	4709	1004	4722	257	4486
	0.5	7871	562	7773	2168	7870	1291	7870	885	7869	**5080**	7867	4145	7848	1081	7845
	0.7	11019	2553	10828	5104	11018	4689	11018	2436	10761	**7842**	9412	7196	9623	3517	10991
nrp-m2	0.3	5099	32	4955	141	4915	93	4765	53	4824	**277**	5095	198	5099	135	4938
	0.5	8499	146	8464	980	8409	1233	8489	336	8369	1816	8497	**2089**	8484	617	8441
	0.7	11898	762	11673	**4158**	11897	2946	11897	1258	11537	3910	10637	3538	9409	1791	11602
nrp-m3	0.3	4140	63	3939	861	4039	866	4072	169	3824	1216	4137	**1462**	4139	305	4099
	0.5	6900	708	6817	988	6899	1294	6899	733	6853	**5663**	6897	4556	6884	1233	6865
	0.7	9660	2381	9131	7163	9659	5423	9659	3269	9326	**9275**	7920	8576	8960	3809	9595
nrp-m4	0.3	4258	24	3968	96	4121	98	4108	69	3987	**338**	4255	124	4251	121	4180
	0.5	7097	172	6954	501	7096	961	7077	245	6903	**2344**	7058	2096	7086	584	7096
	0.7	9936	1509	9768	3872	9935	3418	9935	1471	9502	3625	8082	**4207**	8578	2020	9800

TABLE V
RESULTS: PERCENTAGE OF SOLUTIONS THAT FORM A JOINT PARETO-OPTIMAL FRONT

	Hill climbing				Simulated Annealing				Random	
	First Found Ord		First found Rnd		Ordered		Random			
Data	Point	Spread	Point	Spread	Point	Spread	Point	Spread	Point	Spread
nrp-e1	3,63%	5,88%	6,44%	9,80%	45,15%	39,22%	27,00%	0,00%	18,39%	0,00%
nrp-e2	6,80%	8,33%	1,16%	1,67%	60,09%	66,67%	18,15%	0,00%	14,08%	0,00%
nrp-e3	9,36%	8,11%	2,54%	2,70%	60,83%	70,27%	12,62%	0,00%	14,87%	0,00%
nrp-e4	8,87%	13,04%	1,52%	0,00%	59,95%	67,39%	12,90%	0,00%	17,00%	0,00%
nrp-g1	0,70%	0,00%	7,93%	2,13%	58,32%	70,21%	18,31%	0,00%	15,09%	0,00%
nrp-g2	5,20%	6,67%	6,39%	11,11%	58,78%	57,78%	13,52%	0,00%	16,53%	0,00%
nrp-g3	2,25%	4,65%	10,42%	6,98%	48,06%	53,49%	23,25%	0,00%	16,47%	0,00%
nrp-g4	3,89%	4,76%	2,99%	4,76%	59,32%	61,90%	20,18%	0,00%	14,06%	0,00%
nrp-m1	0,49%	0,00%	8,72%	9,80%	52,37%	54,90%	18,05%	0,00%	20,78%	0,00%
nrp-m2	2,09%	6,25%	5,70%	4,69%	51,46%	54,69%	25,06%	0,00%	16,20%	0,00%
nrp-m3	6,42%	8,51%	5,69%	6,38%	46,82%	42,55%	20,38%	0,00%	21,27%	0,00%
nrp-m4	7,66%	7,84%	4,16%	9,80%	40,24%	31,37%	30,44%	0,00%	18,39%	0,00%

for solving a large scale NRP. Its neighbourhood search procedure which examines all of the adjacent solutions before moving to the next step is the limiting factor because the number of such solutions is too large. Among all the analyzed algorithms this is the only one that was incapable of finding a solution with profit above 0 in several cases. We could also add that Sampling Hill Climbing algorithm proved to be inappropriate for solving a large scale NRP as well as Steepest Ascent even though its neighbourhood search procedure is much faster.

H. Answer to RQ4

Our research did not give conclusive evidence that order of neighbourhood search influences the local search heuristic algorithm in finding the optimal solution in large scale NRP. When evaluating the results in finding the highest profit for limited budget, the Random procedure of neighbourhood search outperformed the Ordered procedure in 19 out of 36 instances for Hill Climbing First Found algorithm, and in 13

out of 36 instances for the Simulated Annealing algorithm. When evaluating the results in percentage of solutions that form joint Pareto-optimal front, the two Hill Climbing instances do not give a straightforward answer to this research question. However, the two Simulated Annealing instances give opposite results for this evaluation criterion. Ordered procedure of neighbourhood search does not only outperform the Random procedure greatly in Point range, but Random procedure gives 0% in Spread range.

VI. Conclusion

This paper examined the appropriateness of using Hill Climbing and Simulated Annealing algorithms and compared their contribution in solving the NRP upon a realistic dataset. We used four instances of Hill Climbing algorithm: Steepest Ascent, First Found - Ordered, First Found - Random and Sampling, two instances of Simulated Annealing: Ordered and Random and the Random Search algorithm as a sanity check. The evaluation of results was done in two ways: finding the highest profit for given budget and finding the percentage of forming the joint Pareto-optimal front. The results we had obtained answered three of our four research questions. We concluded that Simulated Annealing algorithm outperforms the Hill Climbing algorithm in solving realistic large scale NRP. Nevertheless, the Hill Climbing algorithm contributes in forming the joint Pareto-optimal front and should not be rejected entirely. Among four instances of Hill Climbing algorithm we analyzed, Steepest Ascent and Sampling did not justify their usage in solving realistic large scale NRP. The fourth research question remained without a clear answer. Comparison of neighbourhood search procedures showed some interesting results when observing the percentage of forming the joint Pareto-optimal front. Further research of that topic using statistical hypothesis tests are needed to give clear conclusions. Besides examined algorithms, there are some genetic algorithms often used for solving the NRP and there are also emerging some new hybrid algorithms with promising results within the Search Based Software Engineering field. Including these algorithms in our analysis is our intention for future research as well.

Acknowledgment

References

[1] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1129–1137. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277179

[2] J. del Sagrado, I. M. del Aguila, and F. J. Orellana, "Ant colony optimization for the next release problem: A comparative study," in *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, ser. SSBSE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 67–76. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2010.18

[3] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro, "A study of the multi-objective next release problem," in *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*, ser. SSBSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 49–58. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2009.21

[4] M. Harman, J. Krinke, J. Ren, and S. Yoo, "Search based data sensitivity analysis applied to requirement engineering," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ser. GECCO '09. New York, NY, USA: ACM, 2009, pp. 1681–1688. [Online]. Available: http://doi.acm.org/10.1145/1569901.1570126

[5] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone-based multilevel algorithm," *IEEE Transactions on Software Engineering*, vol. 38, pp. 1195–1212, 2012.

[6] A. J. Bagnall, V. J. Rayward-Smith, and I. Whittley, "The next release problem," *Information & Software Technology*, pp. 883–890, 2001.

[7] G. Lu, R. Bahsoon, and X. Yao, "Applying elementary landscape analysis to search-based software engineering," in *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, ser. SSBSE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 3–8. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2010.10

[8] H. Jiang, J. Xuan, and Z. Ren, "Approximate backbone based multilevel algorithm for next release problem," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 1333–1340. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830730

[9] Y. Zhang, M. Harman, A. Finkelstein, and S. Afshin Mansouri, "Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation," *Inf. Softw. Technol.*, vol. 53, no. 7, pp. 761–773, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2011.02.001

[10] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang, "A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making," *Requir. Eng.*, vol. 14, no. 4, pp. 231–245, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1007/s00766-009-0075-y

[11] J. T. de Souza, C. L. Maia, F. G. de Freitas, and D. P. Coutinho, "The human competitiveness of search based software engineering," in *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, ser. SSBSE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 143–152. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2010.25

[12] F. Colares, J. Souza, R. Carmo, C. Pádua, and G. R. Mateus, "A new approach to the software release planning," in *Proceedings of the 2009 XXIII Brazilian Symposium on Software Engineering*, ser. SBES '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 207–215. [Online]. Available: http://dx.doi.org/10.1109/SBES.2009.23

[13] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis, "Search based approaches to component selection and prioritization for the next release problem," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, ser. ICSM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 176–185. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2006.56

[14] Y. Zhang and M. Harman, "Search based optimization of requirements interaction management," in *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, ser. SSBSE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 47–56. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2010.16

[15] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, pp. 243–253, 2004.

[16] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," June 2012. [Online]. Available: http://promisedata.googlecode.com

[17] S. Luke, *Essentials of Metaheuristics*. Lulu, 2009, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[18] M. Harman and S. A. Mansouri, "Search based software engineering: Introduction to the special issue of the ieee transactions on software engineering," *IEEE Trans. Software Eng.*, vol. 36, no. 6, pp. 737–741, 2010.

[19] D. Kreher and D. Stinson, *Combinatorial Algorithms: Generation, Enumeration, and Search*, ser. CRC Press series on Discrete mathematics and its applications. Taylor & Francis, 1998. [Online]. Available: http://books.google.hr/books?id=-Lz3BlxzsLoC