Transformed Search Based Software Engineering: A New Paradigm of SBSE

He Jiang(✉), Zhilei Ren, Xiaochen Li, and Xiaochen Lai

School of Software, Dalian University of Technology, Dalian, China
{jianghe, zren, laixiaochen}@dlut.edu.cn,
lil989@mail.dlut.edu.cn

Abstract. Recent years have witnessed the sharp growth of research interests in Search Based Software Engineering (SBSE) from the society of Software Engineering (SE). In SBSE, a SE task is generally transferred into a combinatorial optimization problem and search algorithms are employed to achieve solutions within its search space. Since the terrain of the search space is rugged with numerous local optima, it remains a great challenge for search algorithms to achieve high-quality solutions in SBSE. In this paper, we propose a new paradigm of SBSE, namely Transformed Search Based Software Engineering (TSBSE). Given a new SE task, TSBSE first transforms its search space into either a reduced one or a series of gradually smoothed spaces, then employ search algorithms to effectively seek high-quality solutions. More specifically, we investigate two techniques for TSBSE, namely search space reduction and search space smoothing. We demonstrate the effectiveness of these new techniques over a typical SE task, namely the Next Release Problem (NRP). The work of this paper provides a new way for tackling SE tasks in SBSE.

Keywords: Search based software engineering · Search space transformation · Search space reduction · Search space smoothing · Next release problem

1 Introduction

Since Harman and Jones proposed the conception of Search Based Software Engineering (SBSE) in 2001 [1], SBSE has attracted a great amount of research interests from the society of Software Engineering (SE). As shown in the SBSE repository[1], up to Feb. 3, 2015, 1389 relevant research papers involving over 659 authors around the world have been published.

As stated in [1, 2], a SE task in SBSE is firstly transferred into an optimization problem for solving and then various search algorithms are employed to seek solutions within its search space, a high-dimensional rugged space consisting of points (solutions). Some typical search algorithms include Evolutionary Algorithms (EA, e.g., Genetic Algorithms, Genetic Programming, and Memetic Algorithms), Ant Colony Algorithms (ACO), Tabu Search (TS), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Hill Climb (HC), etc. Up to now, SBSE has covered most SE tasks across all the

[1] SBSE repository: http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

M. Barros and Y. Labiche (Eds.): SSBSE 2015, LNCS 9275, pp. 203–218, 2015.
DOI: 10.1007/978-3-319-22183-0_14

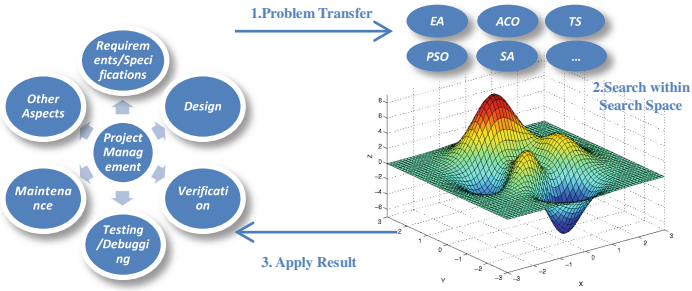

Fig. 1. Roadmap of SBSE

stages of the software lifecycle, including requirement/specification, design, verification, testing/debugging, maintenance, and software project management (see Fig. 1).

Since the search spaces in SBSE are usually rugged with numerous local optima, search algorithms are apt to get trapped into poor local optima. In this paper, we propose a new paradigm of SBSE named Transformed Search Based Software Engineering (TSBSE) to tackle this challenge. In TSBSE, the search spaces are transformed so as to either constrain search algorithms within promising regions or provide better initial solutions for search algorithms. More specifically, we present two techniques for search space transformation, namely search space reduction and search space smoothing. Taking the Next Release Problem (NRP) as a case study, we investigate how to resolve SE tasks within TSBSE.

The remainder of this paper is structured as follows. In Sect. 2, we present the new SBSE paradigm TSBSE. Then in Sect. 3 we present the related work of the NRP. In Sects. 4 and 5, we present the detailed technique of search space smoothing over NRP and the experimental results, respectively. In Sect. 6, we discuss the threats to validity. Finally, we conclude this paper and discuss the future work in Sect. 7.

2 Transformed Search Based Software Engineering

In this section, we introduce the new paradigm of SBSE, namely Transformed Search Based Software Engineering (TSBSE). A key challenge lying in SBSE is that search algorithms in SBSE may easily get trapped into poor local optima, due to the rugged terrain of search spaces with numerous local optima. Therefore, TSBSE aims to tackle the above challenge by transforming the search spaces. As shown in Fig. 2, given a SE task, TSBSE firstly transfers it into a combinatorial optimization problem. Then, TSBSE transforms the related search space to facilitate the process of searching solutions. More specifically, two techniques are available for search space transformation, namely search space reduction and search space smoothing. Third, search algorithms, e.g., EA, ACO, TS, SA, PSO, are employed to search within the transformed search space. In the following part, we illustrate more details of search space reduction and search space smoothing.

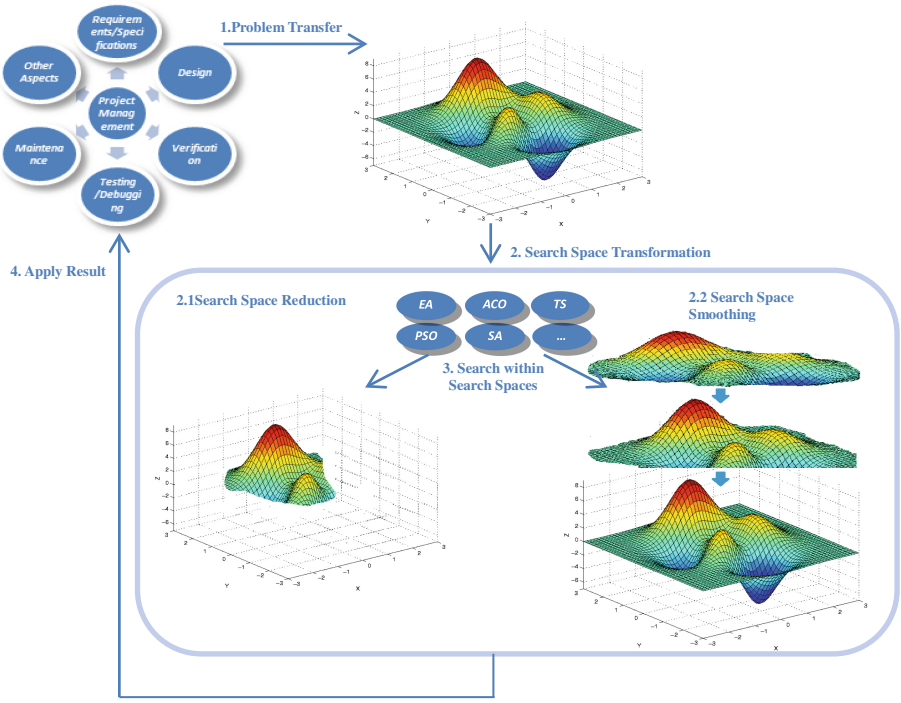

Fig. 2. Roadmap of TSBSE

Search Space Reduction. In SBSE, a search space may consist of numerous local optima and, search algorithms are apt to get trapped into poor local optima. The basic idea of search space reduction in TSBSE is to constrain search algorithms in a reduced search space consisting of high-quality solutions. In such a way, search algorithms could better find high-quality solutions within reasonable running time.

Some related studies [3, 4] in the literature can be viewed as the applications related to search space reduction. For example, in [3], Xuan et al. proposed a backbone based multilevel algorithm to solve NRP. They constrain a shared common part of optimal solutions and reduce the search space into smaller ones. Then, they employ Simulated Annealing to iteratively search for high-quality solutions.

In Fig. 3, we present the pseudo code of the search space reduction. First, in each reduction level, the search algorithm is applied on the current reduced search space to produce a set of high quality solutions $\Gamma_k{}'$. Then, the search space is reduced according to the solutions $\Gamma_k{}'$, e.g., by fixing the common parts of the solutions (lines 2–6 of Algorithm 1). Second, after the search space reduction phase, the search algorithm is applied on the final reduced search space, to obtain a local optimum $\Gamma_{\delta+1}{}'$ (line 7 of Algorithm 1). Third, the local optimum $\Gamma_{\delta+1}{}'$ is transferred gradually back to the feasible solution to the original search space (lines 8–10 of Algorithm 1). During the refinement procedure, the best solution achieved so far is recorded. After the refinement phase, the best solution is returned (line 11 of Algorithm 1).

Algorithm 1: Search Space Reduction

Input: search space Π, search algorithms A, maximum number α of reduction levels, a set of solutions Γ
Output: best solution

```
1  begin
2      for k = 1 to α do
3          Obtain a set of solutions Γ_k by A in Π_k
4          Calculate high quality part Γ_k' of Γ_k
5          Reduce the search space to Π_{k+1} by Γ_k'
6      end
7      Obtain a local optimum Γ_{α+1}' in the reduced search space Π_{α+1} by A
8      for k =α to 1 do
9          Refine the solution with Γ_{k+1}' and Γ_k' in level k
10     end
11     return the best solution achieved
12 end
```

Fig. 3. Pseudo Code of Search Space Reduction

Search Space Smoothing. In SBSE, the terrain of a search space is usually rugged with many poor local optima. Hence, the solutions of search algorithms may heavily depend on the initial solutions fed into search algorithms. For example, Hill Climbing may easily get trapped into a poor local optimum, if it is initialized with a random solution. The idea of search space smoothing is to transform a search space into a series of gradually smoothed ones. In the most smoothed search space, search algorithms are apt to achieve high-quality solutions. Then, the resulting solutions are fed into the second smoothed search space as its initial solutions. Since the terrains of the two search spaces are similar in shape, these initial solutions could lead search algorithms to better hit new high-quality solutions. In such a way, we eventually return to the original search space and achieve the final solutions.

Figure 4 presents the process of search space smoothing over a one-dimensional search space. The original search space is smoothed into two smoothed search spaces. First, a solution is initialized in the most smoothed search space (smoothed search

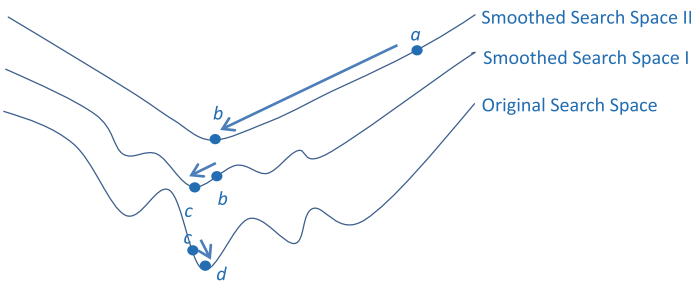

Fig. 4. Illustration of search space smoothing over one-dimensional search space

Algorithm 2: Search Space Smoothing

Input: search space Π, search algorithms A, maximum number β of smoothing levels, a set of solutions Γ
Output: best solution

```
1 begin
2     Generate a smoothed search space Π_0
2     Generate initial solutions Γ_0 in Π_0
3     for k = 1 to β do
4         Tune the search space to Π_k, towards the original, rugged space.
5         Assign the current best solutions Γ_{k-1} as the initial solution
6         Apply A with Γ_{k-1} in Π_k to get the current best solutions Γ_k
7     end
8     return the best solution achieved
9 end
```

Fig. 5. Pseudo Code of search space smoothing

space II) and a local optimum b is achieved. Then, the solution b is used as the initial solution in smoothed search space I, and a local optimum c can be achieved. Finally, the solution c is used as the initial solution in the original search space and the final solution d is eventually returned.

In Fig. 5, we present the pseudo code of the search space smoothing. First, we generate a smoothed search space $\prod_0$, in which the initial solutions Γ_0 is generated (lines 1–2 of Algorithm 2). Then, the search is conducted over a series of search spaces, which are transferred gradually back towards the original, rugged search space (lines 3–7 of Algorithm 2). More specifically, at each iteration, the search space is firstly tuned, and the best solutions up to the previous iteration is regarded as the initial solutions. The algorithm is then applied on the current tuned search space with these initial solutions to obtain the current best solutions. Finally, after the search space is transferred to the original search space, the best solution in the original search space is returned (line 8 in Algorithm 2).

Some early studies [5] in the literature demonstrate that combinatorial optimization problems could be better solved by search space smoothing. However, as to our knowledge, no related work has been done in SBSE. Since for most tasks in SBSE, it is still a challenge on how to prevent search algorithms from getting trapped into poor local optima. We believe that search space smoothing is a promising technique to solve the above problem and may significantly improve the effectiveness of search algorithms in SBSE.

3 Related Work

3.1 The Next Release Problem

Bagnall et al. [6] first proposed the Next Release Problem (NRP) to balance the profits of customers and the developing costs of requirements in the next release of software

systems. Besides customer profits, a variety of problem objectives have been proposed in literature, such as component prioritization [7], fairness [8], etc. According to the number of problem objectives, we can classify the NRP into two categories, namely single-objective NRP (or the NRP for short) and multi-objective NRP (or MONRP for short).

For the category of single-objective NRP, Bagnall et al. [6] apply numerous search-based algorithms, including greedy algorithms, local search, etc., on five randomly generated instances to solve the NRP. In this work, they model the problem as searching for the maximum profits from customers within a predefined cost bound of a software system. The problem can be formalized as follows [6]:

$$\text{Maximize} \sum\nolimits_{i \subseteq S} w_i \text{ subject to } \text{cost}\left(\cup_{i \in S} \hat{R}_i\right) \leq B, B \leq Z^{+} \tag{1}$$

where S is a set of customers, $\hat{R}$ is a set of requirements, w_i is the importance of the ith customer, and $\text{cost}\left(\cup_{i \in S} \hat{R}_i\right)$ means the cost of satisfying all the requirements $\hat{R}$ of the ith customer. The cost should be within some bounds B.

Following the problem definition, Greer and Ruhe [9] propose a genetic algorithm-based approach to iteratively generate the final decision of the NRP. Jiang et al. [10] propose an ant colony optimization algorithm with a local search operator (first found hill climbing) to approximately solve the NRP. A backbone-based multilevel algorithm is proposed in [3]. In this paper, Xuan et al. iteratively reduce the search space by adding the common part of the customers and customers with zero cost to the requirements selection into the combined backbone (approximate backbone and soft backbone). Then they refine the final decision according to the solution in the reduced search space and the combined backbone. Baker et al. [7] extend the NRP with the component selection and ranking, and explore both greedy and simulated annealing algorithms to this problem. Moreover, Ngo-The and Ruhe [11] propose a two-phase optimization approach, which combine integer programming and genetic programming, to allocate the resources of software releases. Paixão et al. proposed a recoverable robust approach for [12] and extands the NRP with a novel formulation which considers the production of robust solutions [13, 14]. Fuchshuber et al. [15] modify the hill climbing algorithm with some patterns observed from the terrain visualization. Araújo et al. [16] draw machine learning models into the NRP. Harman et al. [17] analyze the NRP from the perspective of requirement sensitivity analysis. In this paper, we propose the framework of search space transformation for SBSE, and take the single-objective NRP as a case study. In contrast to solving the problems directly, we smooth the search space to improve the search ability of existing algorithms.

For the category of multi-objective NRP, Zhang et al. [18] first take multi-factors in requirements engineering into consideration and apply the genetic algorithm-based multiobjective optimization to the MONRP. Many related work extend MONRP by balancing factors between the benefits and fairness [19], sensitivity [20] robustness [21], or uncertainty [22]. Besides, Saliu and Ruhe [23] aim at optimizing release plans from both the business perspectives and the implementation perspectives. Zhang et al. [24] seek to balance the requirements needs of today with those of the future. Veerapen et al. [25] evaluate integer linear programming approach on both the single-objective

and multi-objective NRP. A recent work by Zhang et al. [26] conduct comprehensive empirical study of different search algorithms across different real world datasets in NRP. Another review is conducted by Pitangueira [27].

3.2 Search Space Reduction for the NRP

Search space reduction is an effective approach to search high quality solutions for SBSE in the framework of search space transformation. A typical application of search space reduction has been studied in [3].

In [3], Xuan et al. propose a Backbone-based Multilevel Algorithm (BMA) to solve the NRP. The BMA employs multilevel reductions to iteratively reduce the problem scale and refine the final optimal solution. For each level, BMA combines approximate backbone with soft backbone to build a part of final optimal solution and reduce the search space by removing the common part of the optimal customers. The approximate backbone is employed to fix the common part of local optima of several local search operators. While the soft backbone is employed to augment the approximate backbone by adding the customers who provide profits with zero cost to the requirements selection. Based on the backbones, BMA resolves the large scale problem to a small one and search solution to it efficiently. Finally, BMA constructs a solution to the original instance by combining the approximate backbone, the soft backbone, and the current solution to the reduced instance together. The experiments show that search space reduction with backbones can significantly reduce the problem scale, meanwhile improves the quality of solutions for NRP without time cost.

4 Search Space Smoothing for the NRP

In this section, we present the main idea of search space smoothing for the NRP, and propose the algorithm framework. More specifically, we first introduce the motivation of search space smoothing. Then, we demonstrate how to realize the search space smoothing framework.

4.1 The Motivation

The motivation of search space smoothing is intuitive and simple, which is usually described analogously as “seeing the forest before trees” [28]. The idea of search space smoothing is to capture the general characteristics of the search space first, and gradually gain more details of the search terrain. This process is realized by transferring the search terrain from a smooth on towards the original rugged one. To achieve the performance improvement with search space smoothing, researchers have proposed various approaches. Among these approaches, most adopt the instance perturbation techniques, to realize the gradual transfer from smooth search terrains to the original rugged ones. Through a series of instance perturbations, search space smoothing intends to avoid the search from being stuck by locally optimal traps. With the help of well-defined smoothing strategies, search space smoothing is able to conduct such

search space transferring at the cost of only a few extra parameters. In the existing literatures, there exist several smoothing approaches, such as power law smoothing, sigmoidal smoothing, etc. In this study, we take the power law smoothing as an example, to investigate the possibility of realizing the search space smoothing.

As mentioned in Sect. 2, the objective of the NRP is to maximize the revenue of the selected customer subset. Following the existing search space smoothing studies [5, 28], we propose the following smoothing formula:

$$w_i(\alpha) = \begin{cases} \bar{w} + (w_i' - \bar{w})^{\alpha}, & w_i' \geq \bar{w} \\ \bar{w} - (\bar{w} - w_i')^{\alpha}, & w_i' < \bar{w} \end{cases}, \tag{2}$$

where w' is the normalized revenue of the ith customer, $\bar{w}$ indicates the normalized average revenue of all the customers, and α is a parameter that controls the degree of smoothing. By introducing the parameter, smoothed instances could be generated. In Fig. 6, we provide the illustration of the influence caused by the smoothing parameter. In the figure, the x- and y- axes represent the normalized revenue and the smoothed revenue calculated with Eq. 2, respectively. The curves in the figure correspond to different configurations of the parameter. It is obvious that when $\alpha \gg 1$, all the revenues tend to be equal, meanwhile when α is 1, the instance would degenerate to the original instance. By adaptively controlling α, the search terrain could be fine-tuned accordingly.

In Fig. 7, we present the pseudo code of the search space smoothing framework for the NRP. The framework works in an iterative paradigm, according to certain schedule of the parameter α. For example, suppose the simplest schedule: let α decreases linearly from 5 to 1. At each iteration, we first construct a smoothed instance according to Eq. 2 (line 7 of Algorithm 3). Then, with the best solution up to the previous iteration as the initial solution, we apply the embedded algorithm to improve the incumbent solution (lines 8–9 of Algorithm 3). As α decreases towards 1, the search space gets transferred towards the original space. Finally, when the main loop terminates, the best solution achieved is returned (line 11 of Algorithm 3).

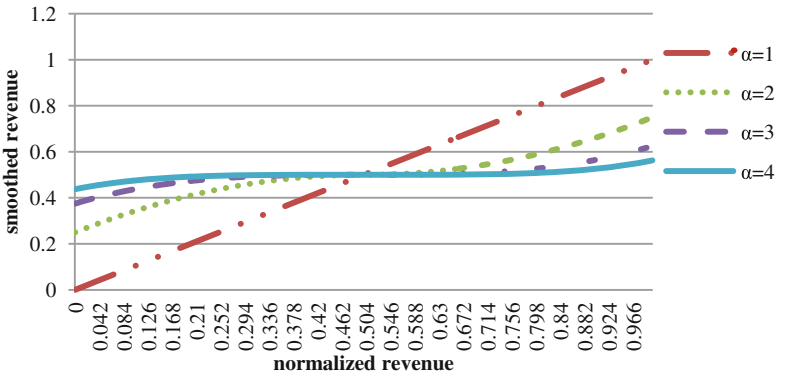

Fig. 6. Revenue Smoothing Transformation Scheme

Algorithm 3: Search Space Smoothing for NRP

Input: Embedded Algorithm *A*
Output: optimized solution s

```
1  begin
2      for each customer i do
3          Normalize all the revenues so that 0≤ ω_i' ≤ 1
4      end
5      Generate initial solutions
6      for α in predefined schedule do
7          Set the revenue vector with respect to Eq. 2
8          Assign the initial solution with the current best solution
9          Apply A with the smoothed instance for optimization
10     end
11     return best solution achieved
12 end
```

Fig. 7. Pseudo Code of the search space smoothing framework for NRP

We could observe that, search space smoothing does not make assumptions about the algorithm which is embedded in the framework. Hence, it is easy to implement search space smoothing based variants that adopt other algorithms. For the following section, we would examine the flexibility of search space smoothing with a simple evolutionary search algorithm.

4.2 Search Space Smoothing Based Memetic Algorithm

After introducing the background information of the search space smoothing framework, we proceed to adapt the smoothing techniques for solving the NRP. In this subsection, we embed a simple Memetic Algorithm (MA) into the search space smoothing framework (denoted as SSS-MA). The reason we choose MA as the embedded algorithm is that, MA could be viewed as the combination of genetic algorithm and local search techniques. By balancing the intensification ability of local search and the diversification of the genetic operators, MAs have achieved promising performances in various problem domains [29, 30].

The pseudo code of SSS-MA is presented in Fig. 8. Similar to the existing genetic algorithms, MA is a population based iterative process. The population consists of a set of solutions to the NRP instance, each of which is encoded as a Boolean vector. MA realizes the problem solving procedure with two phases, i.e., the initialization phase and the main loop phase. First, all the individuals are randomly initialized and evaluated (line 2 of Algorithm 4). Then, for the second phase, the population is iteratively evolved to optimize the individuals. At each iteration, we first modify the parameter α if necessary. In this study, we consider a simple schedule, i.e., decrease α linearly from 5 to 1. After the smoothed instance is constructed (lines 4–5 of Algorithm 4), genetic operators such as crossover and mutation are applied over each individual. In this

Algorithm 4: Search Space Smoothing based Memetic Algorithm for NRP

Input: maximum iterator n*Iter,* population size n*Pop*, elitism rate *eRate*, mutation rate m*Rate*
Output: best solution achieved

```
1  begin
2      initialization
3      for i ← 1 to nIter do
4          α ← ⌊6 − (5 × i)/nIter⌋
5          Modify instance variables with Eq. 2
6          for nPop × (1 − eRate) do
7              Randomly select two individuals as parents
8              Apply uniform crossover
9              Apply bit-flipping mutation over the offspring
10             Apply hill climbing over the offspring
11         end
12         Apply elitism selection
13     end
14     return best solution achieved
15 end
```

Fig. 8. Pseudo Code of SSS-MA

study, uniform crossover and bit-flipping mutation are employed to produce the offspring individuals (lines 7–9 of Algorithm 4). Furthermore, in addition to the genetic operators, MA features the use of local search operators (line 10 of Algorithm 4). In this study, we apply a bit-flipping based hill climbing procedure as the local search operator. After all the operators have been applied, all the individuals and their offspring undergo a selection operator, to construct the population for the next iteration (line 12 of Algorithm 4). In this study, the truncation based selection mechanism is adopted. With the selected individuals, the evolution process continues the following iterations, until certain stopping criteria are met.

5 Experiments

In this section, we present the extensive experiments, to demonstrate the effectiveness of search space smoothing applied to the NRP. More specifically, we first present the preliminary information of the experiments. Then, numerical experiments are conducted over the benchmark instances. We compare the SSS-MA with the baseline MA, to examine the performance of the proposed algorithm. Finally, we investigate why search space smoothing works by illustrate the anytime performance of SSS-MA.

Before presenting the experimental results, we first briefly give the background information of the experiments. In this study, the algorithms are implemented in C++, compiled with g++ 4.9. The experiments are conducted on a PC with an Intel Core i5

3.2 GHz CPU and 4 GB memory, running GNU/Linux with kernel 3.16. For the benchmark instances, there are two classes from [3] and [6], respectively.

To evaluate the performance of SSS-MA, we consider two comparative algorithms. First, we adopt the basic MA as the baseline algorithm. The only difference between MA and SSS-MA lies in the smoothing mechanism. By comparing the two algorithms, we are able to examine the usefulness of search space smoothing. Second, besides MA, we employ the solution achieved by the backbone guided algorithm BMA [3] as the reference to evaluate the effectiveness of SSS-MA objectively, since BMA is among the best heuristics for the NRP. Next, since there are parameters in both MA and SSS-MA, we have to conduct the parameter tuning task. In this study, we choose to tune the elitism ratio and the mutation rate, and fix the rest parameters for the two algorithms. The reason for this experiment scheme is that, during the implementation, we find that these two parameters have the major influence on the performance. In particular, we employ the automatic tuning tool irace [31]. The parameter settings for the algorithms are summarized in Table 1.

Table 1. Parameter setup for MA and SSS-MA

Parameter	MA	SSS-MA
Maximum iteration	5000	5000
Population size	10	10
Elitism rate	0.51	0.27
Mutation rate	0.02	0.01

5.1 Numerical Results

After the preliminary experiment, we proceed to carry out the numerical experiments. For each benchmark instances, we independently execute the two algorithms for 10 times, and report the results in Table 2. The table is organized as follows. The first column indicates the instances. The second column presents the best known solution quality achieved by BMA. Then, in columns 3–5 and 6–8, the results for MA and SSS-MA are given, respectively. For each algorithm, we list the maximum and the mean of the solution quality, as well as the average time in seconds. From the table, several interesting observations could be drawn. First, from the effectiveness aspect, SSS-MA is able to achieve solutions with better quality than SSS-MA. Over the 39 instances, SSS-MA outperforms MA over 36 instances, in terms of the best solution quality. When we compare the average solution quality of the two algorithms, similar observations could be found. For both the two comparison scenarios, the conclusion that SSS-MA outperforms MA is supported by the nonparametric Wilcoxon's two-sided signed rank test (with p-values < 0.0001). In particular, SSS-MA obtains solutions that are better than the currently best known solutions over 6 instances. Second, from the efficiency aspect, SSS-MA is slower than MA over all the instances. The reason for this phenomenon might be that, for SSS-MA, especially during its beginning iterations, the search is conducted over the smoothed terrain, it is possible

Table 2. Results

Instance	BMA	MA			SSS-MA		
	Best	Best	Average	Time	Best	Average	Time
nrp1-0.3	1201	**1204**	1191.1	1.22	1200	1189.2	2.59
nrp1-0.5	1824	**1836**	1812.8	1.38	1834	1784.2	2.62
nrp1-0.7	2507	2507	2507	1.15	2507	2507	2.42
nrp2-0.3	4726	4007	3927.7	5.57	**4365**	4179.8	13.53
nrp2-0.5	7566	7034	6840.7	7.16	**7353**	7202.2	15.79
nrp2-0.7	10987	10585	10419	7.85	**10683**	10589.5	16.56
nrp3-0.3	7123	6846	6756	7.25	**7001**	6894.2	14.70
nrp3-0.5	10897	10566	10522.2	7.95	**10758**	10644.6	15.75
nrp3-0.7	14180	13867	13819.5	7.78	**13990**	13953	15.58
nrp4-0.3	9818	8950	8841.6	17.96	**9164**	9003.8	29.72
nrp4-0.5	15025	14609	14457.6	20.22	**14794**	14613.6	32.95
nrp4-0.7	20853	19996	19906.6	22.60	**20205**	20117.4	35.76
nrp5-0.3	17200	14873	14564.3	19.33	**15417**	15165.7	40.68
nrp5-0.5	24240	22409	22204.5	14.95	**22785**	22616.3	34.89
nrp5-0.7	28909	27494	27283.6	10.41	**27854**	27761.8	28.75
nrp-e1-0.3	7572	7396	7344.5	12.95	**7539**	7460.8	20.63
nrp-e1-0.5	10664	10607	10555	15.16	**10740**	10676.3	22.90
nrp-e2-0.3	7169	7053	6984.2	15.04	**7097**	7046.2	22.16
nrp-e2-0.5	10098	10021	9964.8	17.93	**10081**	10021.7	25.27
nrp-e3-0.3	6461	6345	6305	10.00	**6385**	6329.4	16.59
nrp-e3-0.5	9175	9090	9034.5	11.55	**9095**	9054.2	18.35
nrp-e4-0.3	5692	5553	5525.1	10.66	**5633**	5576.7	16.24
nrp-e4-0.5	8043	7982	7919	12.46	**7989**	7965.4	18.14
nrp-m1-0.3	10008	9573	9490.6	17.31	**9735**	9627	28.26
nrp-m1-0.5	14588	14416	14305.7	20.38	**14607**	14470.4	32.44
nrp-m2-0.3	8272	8044	7927.6	16.70	**8128**	8030.6	25.49
nrp-m2-0.5	11975	11970	11879.1	20.28	**12045**	11979.7	29.43
nrp-m3-0.3	9559	9302	9226.6	15.22	**9470**	9332	26.40
nrp-m3-0.5	14138	14123	14045.9	17.85	**14289**	14167.7	30.00
nrp-m4-0.3	7408	7197	7123.8	13.85	**7288**	7211.5	21.87
nrp-m4-0.5	10893	10836	10774.7	16.53	**10940**	10875.1	24.92
nrp-g1-0.3	5938	5917	5862	9.24	**5930**	5897.4	15.04
nrp-g1-0.5	8714	8657	8610.4	11.00	**8701**	8669.5	17.08
nrp-g2-0.3	4526	4474	4452.2	8.34	**4495**	4477.1	12.42
nrp-g2-0.5	6502	6447	6436.4	9.91	**6489**	6455.5	14.09
nrp-g3-0.3	5802	**5749**	5722.7	8.50	5739	5711.9	14.37
nrp-g3-0.5	8402	8327	8293.7	9.77	**8359**	8308.9	15.90
nrp-g4-0.3	4190	4149	4134.9	6.88	**4173**	4143.2	10.79
nrp-g4-0.5	6030	6002	5977.4	7.86	**6010**	5977.8	11.93

Fig. 9. Performance Comparison between MA and SSS-MA

that the hill climbing operator may be more time consuming. However, we can see that the times for the two algorithms are in the same order of magnitude.

5.2 Anytime Performance Comparison

In the previous subsection, we have observed that SSS-MA outperforms MA in terms of solution quality. However, SSS-MA is more time consuming accordingly. In this subsection, we intend to investigate the dynamic characteristics of the two algorithms, by visually comparing their anytime performance. We choose nrp-2-0.3 and nrp-g1-0.5 as the typical instances, and plot the anytime performance curves of MA and SSS-MA. In Fig. 9, the x-axis indicates the number of iterations elapsed, and the y-axis indicates the average solution quality achieved by the two algorithms.

From the figure, we find that for the beginning iterations, MA outperforms SSS-MA. For example, over nrp2-0.3, after 200 iterations, the solution quality of MA is 31559, while that of SSS-MA is 30089. However, as the search terrain gets transferred back to the original terrain, the solution obtained by SSS-MA is improved accordingly. After 400 iterations, SSS-MA is able to achieve better solutions compared to MA. These observations demonstrate that SSS-MA is able to avoid locally optimal traps to some extent. Similar observations could be found over NRP-g1-0.5. Similar phenomenon could be observed on the other instance we examine. Based on the anytime performance comparison, we partially confirm that the reason for the slow convergence of SSS-MA is caused by the smoothing operation.

6 Threats to Validity

In this paper, we demonstrate the effectiveness of search space smoothing, one of the techniques in TSBSE, on a typical SE task the NRP. However, there are some threats to validity: First, we validate the effectiveness of search space smoothing in TSBSE on 39 instances in the NRP and demonstrate the effectiveness of search space reduction with several related work. The proposed technique should be validated with more real world

data set and SE tasks. Second, we smooth the search space in the NRP with a typical search space smoothing technique, the power law smoothing, which has been successfully applied in several research work [28, 29]. With this technique, we improve the optimal solution of the NRP with some time costs. However, the search space smoothing technique may slightly affect the results of our case study. In the future, we should validate and compare more search space smoothing techniques for SE tasks, and propose more time efficient formulas.

7 Conclusion and Future Work

In this paper, we address the conception of Transformed Search Based Software Engineering (TSBSE). Taking the Next Release Problem from the requirements engineering as a case study, we investigate the feasibility of applying search space smoothing for SBSE. The contributions of this study are tri-fold. First, we propose the conception of the TSBSE, which unifies the techniques such as search space reduction and search space smoothing. To the best of our knowledge, this is the first time such conception is issued in the software engineering community. Second, we develop a Search Space Smoothing based Memetic Algorithm (SSS-MA). We demonstrate that, with minor modification, algorithms could be embedded into the search space smoothing framework. Furthermore, numerical results reveal that, the proposed algorithm is able to update several best known solutions over the benchmark of the NRP instances. For the future work, we are interested in the following directions. First, SSS-MA tends to be slower than directly executing the embedded algorithm. Hence, how to accelerate the problem solving process deserves more efforts. Second, in the existing literature, there exist several smoothing schemes. Comparisons between these schemes seem interesting. Third, we would explore the possibility of extending the search space smoothing framework to more problems in software engineering.

Acknowledgement. This work is supported in part by the National Natural Science Foundation of China under Grants 61175062, 61370144, and 61403057, and in part by China Postdoctoral Science Foundation under Grant 2014M551083.

References

1. Harman, M., Jones, B.: Search-based software engineering. Inf. Softw. Technol. **43**(14), 833–839 (2001)
2. Harman, M., Mansouri, A., Zhang, Y.: Search based software engineering: trends, techniques and applications. ACM Comput. Surv. **45**(1), 11–75 (2012). Article 11
3. Xuan, J., Jiang, H., Ren, Z., Luo, Z.: Solving the large scale next release problem with a backbone-based multilevel algorithm. IEEE TSE **38**(5), 1195–1212 (2012)
4. Ren, Z., Jiang, H., Xuan, J., Luo, Z.: An accelerated limit crossing based multilevel algorithm for the p-Median problem. IEEE TSMCB **42**(2), 1187–1202 (2012)
5. Jun, G., Huang, X.: Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). IEEE Trans. Syst. Man Cybern. **24**(5), 728–735 (1994)

6. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Inf. Softw. Technol. **43**(14), 883–890 (2001)
7. Baker, P., Harman, M., Steinhofel, K., Skaliotis, A.: Search based approaches to component selection and prioritization for the next release problem. In Software Maintenance, pp. 176–185 (2006)
8. Finkelstein, A., Harman, M., Mansouri, S.A., Ren, J., Zhang, Y.: A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requirements Eng. **14**(4), 231–245 (2009)
9. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. **46**(4), 243–253 (2004)
10. Jiang, H., Zhang, J., Xuan, J., Ren, Z., Hu, Y.: A hybrid ACO algorithm for the next release problem. In: SEDM, pp. 166–171 (2010)
11. Ngo-The, A., Ruhe, G.: Optimized resource allocation for software release planning. IEEE Trans. Software Eng. **35**(1), 109–123 (2009)
12. Paixão, M.H.E., de Souza, J.T.: A recoverable robust approach for the next release problem. In: Ruhe, G., Zhang, Y. (eds.) SSBSE 2013. LNCS, vol. 8084, pp. 172–187. Springer, Heidelberg (2013)
13. Paixão, M., Souza, J.: A scenario-based robust model for the next release problem. In: GECCO, pp. 1469–1476 (2013)
14. Paixão, M., Souza, J.: A robust optimization approach to the next release problem in the presence of uncertainties. J. Syst. Softw. **103**, 281–295 (2014)
15. Fuchshuber, R., de Oliveira Barros, M.: Improving heuristics for the next release problem through landscape visualization. In: Le Goues, C., Yoo, S. (eds.) SSBSE 2014. LNCS, vol. 8636, pp. 222–227. Springer, Heidelberg (2014)
16. Araújo, A.A., Paixão, M.: Machine learning for user modeling in an interactive genetic algorithm for the next release problem. In: Le Goues, C., Yoo, S. (eds.) SSBSE 2014. LNCS, vol. 8636, pp. 228–233. Springer, Heidelberg (2014)
17. Harman, M., Krinke, J., Medina-Bulo, I., Palomo-Lozano, F., Ren, J., Yoo, S.: Exact scalable sensitivity analysis for the next release problem. TOSEM **23**(2), 19 (2014)
18. Zhang, Y., Harman, M., Mansouri, S.A.: The multi-objective next release problem. In: GECCO, pp. 1129–1137. ACM (2007)
19. Finkelstein, A., Harman, M., Mansouri, S.A., Ren, J., Zhang, Y.: A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requirements Eng. **14**(4), 231–245 (2009)
20. Harman, M., Krinke, J., Ren, J., Yoo, S.: Search based data sensitivity analysis applied to requirement engineering. In: GECCO, pp. 1681–1688. ACM (2009)
21. Gueorguiev, S., Harman, M., Antoniol, G.: Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In: GECCO, pp. 1673–1680. ACM (2009)
22. Li, L., Harman, M., Letier, E., Zhang, Y.: Robust next release problem: handling uncertainty during optimization. In: GECCO, pp. 1247–1254 (2014)
23. Saliu, M.O., Ruhe, G.: Bi-objective release planning for evolving software systems. In: FSE, pp. 105–114 (2007)
24. Zhang, Y., Alba, E., Durillo, J.J., Eldh, S., Harman, M.: Today/future importance analysis. In: GECCO, pp. 1357–1364. ACM (2007)
25. Veerapen, N., Ochoa, G., Harman, M., Burke, E.K.: An integer linear programming approach to the single and bi-objective next release problem. Inf. Softw. Technol. **65**, 1–13 (2015)
26. Zhang, Y., Harman, M., Ochoa, G., Ruhe, G., Brinkkemper, S.: An empirical Study of meta-and hyper-heuristic search for multi-objective release planning. RN **14**, 07 (2014)

27. Pitangueira, A.M., Maciel, R.S.P., Barros, M.: Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. J. Syst. Softw. **103**, 267–280 (2014)
28. Coy, S.P., Golden, B.L., Runger, G.C., Wasil, E.A.: See the forest before the trees: fine-tuned learning and its application to the traveling salesman problem. IEEE SMCA. **28**, 454–464 (2014)
29. Fraser, G., Arcuri, A., McMinn, P.: A Memetic Algorithm for whole test suite generation. J. Syst. Softw. **103**(2), 311–327 (2014)
30. Moscato, P., Cotta, C., Mendes, A.: Memetic algorithms. In: Onwubolu, G.C., Babu, B.V. (eds.) New Optimization Techniques in Engineering, pp. 53–85. Springer, Berlin, Heidelberg (2004)
31. Lopez-Ibanez, M., Dubois-Lacoste, J., Stutzle, T., et al.: The irace package, iterated race for automatic algorithm configuration. IRIDIA, Universite Libre de Bruxelles, Belgium, Technical Report TR/IRIDIA/2011-004 (2011)