Exact Scalable Sensitivity Analysis for the Next Release Problem

MARK HARMAN and JENS KRINKE, University College London
INMACULADA MEDINA-BULO and FRANCISCO PALOMO-LOZANO, University of Cádiz
JIAN REN and SHIN YOO, University College London

The nature of the requirements analysis problem, based as it is on uncertain and often inaccurate estimates of costs and effort, makes sensitivity analysis important. Sensitivity analysis allows the decision maker to identify those requirements and budgets that are particularly sensitive to misestimation. However, finding scalable sensitivity analysis techniques is not easy because the underlying optimization problem is NP-hard. This article introduces an approach to sensitivity analysis based on exact optimization. We implemented this approach as a tool, OATSAC, which allowed us to experimentally evaluate the scalability and applicability of Requirements Sensitivity Analysis (RSA). Our results show that OATSAC scales sufficiently well for practical applications in Requirements Sensitivity Analysis. We also show how the sensitivity analysis can yield insights into difficult and otherwise obscure interactions between budgets, requirements costs, and estimate inaccuracies using a real-world case study.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*Tools*

General Terms: Algorithms

Additional Key Words and Phrases: Next release problem, sensitivity analysis, requirement engineering

ACM Reference Format:
Mark Harman, Jens Krinke, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Jian Ren, and Shin Yoo. 2014. Exact scalable sensitivity analysis for the next release problem. ACM Trans. Softw. Eng. Methodol. 23, 2, Article 19 (March 2014), 31 pages.
DOI: http://dx.doi.org/10.1145/2537853

1. INTRODUCTION

Selecting a set of requirements for the next release of a software system is a complex and demanding problem. However, making good engineering and business judgments concerning these requirements is crucial because the decision process takes place so early in the development of the next release [Cheng and Atlee 2007]. The problem of choosing the optimal set of requirements to include in the next release of a software system has become known as the Next Release Problem (NRP) [Bagnall et al. 2001; Zhang et al. 2007].

The number of choices increases exponentially in the number of requirements, making this a nontrivial problem. The optimization problem that underlies it is NP-hard. The situation is further complicated by the uncertainties inherent to the decision making process. The choice of requirements for the next release is affected by the

This work was partially funded by the Spanish Ministry of Science and Innovation under the National Program for Research, Development and Innovation, project MoD-SOA (TIN2011-27242). It was also supported by the U.K. Engineering and Physical Sciences Research Council (grant numbers EP/D050863, EP/G060525). Data from the EPSRC-funded portions of this work may be available by contacting Dr. Yoo. Note that intellectual property or other restrictions may prevent the full disclosure of this data.
Corresponding author's email: shin.yoo@ucl.ac.uk.

DOI: http://dx.doi.org/10.1145/2537853

expected cost and revenue that will accrue from the inclusion of each candidate requirement.

Unfortunately, a quantitative assessment of development cost and expected revenue can only be based on estimations at decision time. It is well known that such software engineering estimates can be inaccurate [Shepperd 2007]. Even if the requirements engineer is a perfect decision maker, their decisions might therefore be wrong because of the unavoidable inaccuracy of the estimates upon which they are based. An inaccuracy in the estimate of an attribute of one requirement may have a very different impact on the optimal choice of requirements compared to the same inaccuracy concerning the same attribute but of a different requirement. This raises a crucial question, upon which the entire requirements decision process rests.

> Which estimates have the greatest impact upon the optimal requirements choice for the next release?

By identifying the budget values and requirements choices that have a higher impact on the decision, the decision maker can choose to allocate greater resources to the estimation process for these 'sensitive' inputs. This has the aim of improving the quality of the information available where it counts, making best use of available resources. We term the problem of ordering requirements' attributes according to their impact on the choice of requirements for the next release as the Requirements Sensitivity Analysis (RSA) problem.

Sensitivity applies to an individual attribute of a specific requirement for which small changes in attribute value have disproportionately large effects on the composition of the optimal requirements set. Sensitivity can also apply to an entire project budget for which the value of some attribute of many requirements has a disproportionate effect. We can also focus on the sensitivity of an attribute or budget at specific levels of inaccuracy. For instance, it can happen that a budget is most sensitive within a narrow range of potential estimate inaccuracy or that an attribute is only sensitive above a certain level of inaccuracy. Using sensitivity analysis, we can direct the decision maker to those areas of specific sensitivity and thereby help to inform the decision making process.

The relationship between cost, revenue, and inaccuracy cannot be easily understood without automated decision support, because small changes in the value of a certain attribute can have dramatic consequences on the overall solution. This is one of the reasons why the problem of determining the ideal requirements set is so hard. Our approach to the RSA problem is therefore to identify those attributes and budgets for which the sensitivity is unusually high for a given level of inaccuracy simply because of the particular happenstance of attribute value distribution. Recent work has indicated that Search-Based Optimization of the NRP produces human-competitive results, indicating that these approaches provide a good basis for decision support [de Souza et al. 2010].

In order to identify peculiarly sensitive values, our overall approach to RSA is to use multiple optimizations, each of which models a possible inaccuracy in requirements. In this manner, we follow the one-at-a-time (OAT) approach to sensitivity analysis, used in other areas of sensitive analysis [Saltelli et al. 2000]. However, to the best of our knowledge, we are the first authors to address this problem for requirements analysis.

We compute multiple NRP optimizations. On each occasion we perturb the value of a single attribute of a single requirement in order to mimic the effect of inaccurate estimation. We perturb by moving values both upwards and downwards, to capture the effect of underestimates as well as overestimates. We perform small step size perturbations of 5% from -50% to $+50\%$ of the unperturbed attribute value. Each execution of the NRP optimization algorithm therefore constructs a solution set of

requirements for an instance of the problem in which one estimate was inaccurate by a given amount.

Much of the previous work on optimization algorithms for the NRP and release planning has concerned the application of metaheuristic search algorithms [Bagnall et al. 2001; Baker et al. 2006; Greer and Ruhe 2004]. Metaheuristic algorithms have been favored, partly because they scale well and handle multiple objectives [Saliu and Ruhe 2007a; Zhang et al. 2007]. By contrast, exact algorithms from the Operations Research literature that locate a globally optimal solution have been less widely used. Furthermore, though exact techniques have been previously applied to the NRP and release planning [van den Akker et al. 2005], their scalability has not been explored in the literature on NRP.

In order to attack the RSA problem using multiple executions of search-based optimization, we need to use an exact algorithm. If we use an inexact algorithm, we cannot be sure whether variations in results obtained from original and perturbed NRP problems accrue from inherent sensitivity or merely from the nature of the algorithms used to optimize. However, in choosing to use an exact approach to optimization, we are confronted with the issue of scalability.

In this article, we introduce and evaluate our approach to the RSA problem and a variant of the Nemhauser–Ullmann's algorithm [Nemhauser and Ullmann 1969] for exact optimization of each NRP instance. Since the algorithm must be run $A \times M$ times, where A is the number of attributes to be perturbed and M is the number of perturbations applied, we experimentally study and report on the scalability of the approach to demonstrate that it is practical. We also demonstrate that our approach is useful in practice, presenting three case studies of its use on a real-world RSA problem. The case studies reveal both peculiarly sensitive attributes and sensitive budgets.

The primary contributions of this article are as follows.

—We introduce the exact RSA problem and an OAT solution using the Nemhauser–Ullmann's exact optimization algorithm as a solver for each OAT step.
—We implemented our approach in a tool, OATSAC (One At a Time Sensitivity Analysis for Cost-benefit)[1], and used it to construct an experimental study of the scalability of the approach. The results of the experimental study indicate that the approach scales well for both size and complexity of NRP problems. The size is measured simply in terms of the number of requirements from which the NRP choice has to be made. The search-space size is exponential in this parameter. For complexity, we use the degree of correlation between cost and value, since high correlations usually denote complex (and therefore potentially less scalable) problem instances. The results of the experimental study provide evidence to support the claim that our approach is scalable and therefore practical.
—We report the results of a case study in which the approach was applied to the Motorola NRP requirements set (see Section 6.1 for details). The results show how a decision maker can use our approach to identify anomalous cases, both in terms of requirement attributes that are especially sensitive and also entire budgets for which the decision problem is sensitive. Identification of sensitive budgets allows the decision maker to negotiate for a more stable (less sensitive) budget, thereby addressing business concerns. Identification of attribute sensitivity allows the decision makers to target resources on sensitive attribute estimation, thereby addressing management and engineering concerns. The fact that we found cases of both kinds of sensitivity in our real-world case study provides evidence to support the claim that our approach may be useful in practice.

[1]OATSAC is availble for download at http://ucase.uca.es/nrp.

The organization of the rest of the article is as follows: Section 2 introduces the idea of Requirements Sensitivity Analysis, based on iterative solutions to perturbed NRP instances. Section 3 introduces the tool, OATSAC, which implements our approach and discuses the algorithmic choices involved in producing the exact and scalable results for NRP perturbations. Section 4 presents the research questions, the answers to which are presented in Sections 5 and 6. In Section 7, we extend our analysis to interactions between two requirements estimates (a problem we term the second order interaction problem), and in Section 8, we describe the actionable findings that a decision maker might exploit arising from our case study on the Motorola dataset. Section 9 considers the threats to validity and limitations of our work, while Section 10 presents related work on the NRP and sensitivity analysis. Section 11 discusses implications for subsequent work and Section 12 presents directions for future work. Finally, Section 13 concludes.

2. REQUIREMENTS SENSITIVITY ANALYSIS

Sensitivity analysis (SA) consists of assessing the contribution to the overall uncertainty of a solution that accrues from the individual uncertainty due to some specific element of the solution [Helton et al. 2006]. In this article, we adopt a one-at-a-time (OAT) approach to sensitivity analysis [Saltelli et al. 2000], which is also referred to as Local Sensitivity Analysis [Saltelli et al. 2008] and Nominal Range Sensitivity Analysis [Frey and Patil 2002] in the literature.

OAT methods vary one parameter at a time repeatedly, while all of the other parameters are maintained at their fixed, baseline values. OAT is the most popular SA practice in the literature [Saltelli and Annoni 2010]. Its strengths can be summarized as follows.

—The baseline vector provides a safe starting point from which to generate all perturbed versions. This minimizes the chance of generating invalid inputs.
—OAT guarantees that all impact is solely due to the perturbation in the input, provided that the model does not have a stochastic term.
—OAT does not produce type-I statistical errors; a nonzero impact always implies impacts from the perturbed input.

Hitherto, there has been no previous work on exact SA (OAT or any other form of SA) for the Next Release Problem in Requirements Engineering. This is an important omission in the previous literature, because the choice of requirements for the next release is a decision that is inherently and intrinsically based upon estimates that are widely believed to be unreliable. Nevertheless decisions concerning requirements do have to be taken and these decisions, coming as they do early in the lifecycle, can have a profound effect on the cost and effectiveness of the overall system. As this article shows, using OAT, it is possible to gain insight into those estimates that can have dramatic and otherwise unexpectedly high impact on the choice of requirements.

Our OAT approach is based on multiple iterations of the NRP optimization problem in search-based software engineering (SBSE), a subfield of software engineering that has grown rapidly in recent years [Colanzi et al. 2012; Freitas and Souza 2011; Harman 2007; Harman et al. 2012a, 2012b, 2012c; Zhang et al. 2008]. Each iteration caters for a different perturbed version of the original estimates. The NRP deals with the selection of a subset of requirements based on their desirability, for example, on their total expected revenue, while subject to constraints such as a limited budget [Bagnall et al. 2001]. The original formulation of NRP, due to Bagnall et al., considers an objective function whose value depends on the satisfaction of a set of customers via the inclusion of their demanded requirements in the next release of a complex software product; the aim of the problem being to select the subset of requirements (and, thus, of satisfied

customers) maximizing the value of the aforementioned function without exceeding the company's budget.

More formally, let S be a set containing n requirements. For each requirement $x \in S$, let $cost(x)$ represent its cost, and $revenue(x)$ its revenue. The extension of *cost* and *revenue* to S can be simply defined as follows.

$$cost(S) = \sum_{x \in S} cost(x),$$
$$revenue(S) = \sum_{x \in S} revenue(x).$$

This NRP problem consists of selecting the subset of requirements R with the highest revenue and with cost in budget. Let B the budget, then $R \subseteq S$ is an optimal solution to the NRP problem if $cost(R) \leq B$, and for all $Q \subseteq S$, the following property holds:

$$cost(Q) \leq B \rightarrow revenue(Q) \leq revenue(R).$$

We can achieve this by maximizing $revenue(R)$ subject to $R \subseteq S$ and $cost(R) \leq B$. Let $c = [c_1, \ldots, c_n]$ and $r = [r_1, \ldots, r_n]$ the cost and revenue vectors for the n requirements in S, and $s = [s_1, \ldots, s_n] \in \{0, 1\}^n$ a solution vector, that is, a bitset identifying a subset of S. Then, we obtain the following equivalent optimization problem, which is a binary integer linear program.

$$\begin{aligned} \text{NRP}: \ & \max s \cdot r \\ & \text{subject to} \\ & s \cdot c \leq B \\ & s \in \{0, 1\}^n \end{aligned} \tag{1}$$

It is clear from the preceding discussion that the NRP problem is essentially the classical 0/1 Knapsack Problem (KP). This is an NP-hard problem [Karp 1972]. This formulation assumes that costs and values are additive. If this assumption does not hold (e.g., there are synergies between requirements that reduce costs), then a more complex formulation is required. The exploration of such models and their impact on sensitivity analysis is an interesting topic for future work.

Previous work on solving the NRP problem has focused on metaheuristic optimization: while deterministic optimization techiques such as the greedy heuristic has been used [AlBourae et al. 2006], many have applied stochastic optimisation [Feather and Menzies 2002; Greer and Ruhe 2004; Jalali et al. 2008; Ngo-The and Ruhe 2009; Ruhe and Greer 2003; Ruhe and Ngo-The 2004; Zhang et al. 2008]. However, the inherent randomness in stochastic optimization poses new challenges to sensitivity analysis: if a *what-if* scenario yields a result different from the original instance of NRP, there is no way of knowing whether the difference is due to the stochastic nature of the approach or the impact of the parameter perturbation. Deterministic approximation such as greedy algorithms would eliminate the randomness but still at the cost of the optimality of the solution. Previous work has demonstrated that Greedy approaches are far from optimal in practice for requirements selection tasks [Ruhe et al. 2003].

To guarantee both the optimality of the solution and the lack of inexactness, it is necessary to use an exact algorithm for NRP: solutions obtained by exact algorithms will be optimal. However, this in turn raises the scalability concern, because OAT requires a large number of what-if problem solving experiments to produce a sufficiently detailed sensitivity analysis. Our research question addresses this question of scalability of the OAT approach combined with an exact algorithm.

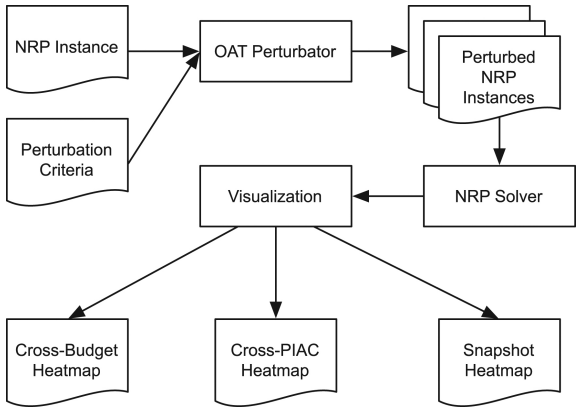


Fig. 1. Overall architecture of tool.

ALGORITHM 1: OAT Sensitivity Analysis Procedure

Input: An NRP instance, P_0, and a set of n perturbation criteria, I
Output: Visualizations of n different scenarios

```
(1)   ▶ Solve the original instance as a reference
(2)   S_0 ← solve(P_0)
(3)   S ← Ø
(4)   ▶ Generate n different perturbations
(5)   P ← Ø
(6)   foreach i ∈ I
(7)       P_i ← apply i to P_0
(8)       P ← P ⋃ {P_i}
(9)   ▶ Solve perturbed problem instances
(10)  foreach P_i ∈ P
(11)      S_i ← solve(P_i)
(12)      S ← S ⋃ {S_i}
(13)  ▶ Visualize the results
(14)  foreach S_i ∈ S
(15)      Visualize the difference between S_i and S_0
```

3. THE OATSAC RSA TOOL

Algorithm 1 outlines the overall procedure of one-at-a-time sensitivity analysis for NRP. Given the original NRP instance, P_0, and a set of predefined perturbations, we generate and solve n different versions of the original instance. The results are then compared to the solution to the original instance for the visualization of the differences.

Figure 1 shows the overall architecture of our OAT sensitivity analysis tool, OATSAC (One At a Time Sensitivity Analysis for Cost-benefit), which consists of three main components: OAT perturbator, NRP solver, and visualization. OAT Perturbator accepts the original problem instance and a set of perturbation criteria as input, and generates a set of perturbed problem instances accordingly. These are fed into NRP Solver. While we use the Nemhauser–Ullmann's exact algorithm, any other NRP solver can be plugged in. The visualization component compares the different results from perturbed problem instances and highlights the impacts in three different types of heat-maps.

We use the term PIAC (percentage of increase in actual cost) to denote an estimate inaccuracy. The PIAC is the degree to which the estimator underestimated the true cost (since the actual cost is increased). A PIAC value of x indicates the the actual cost

is $x\%$ larger than the estimated cost. We allow both positive and negative values for PIAC, so that a negative value indicates an overestimate of cost.

—*Snapshot Heatmap*. These report the impact that a specific perturbation has on all requirements and budgets.
—*Cross-PIAC Heatmap*. These summarize the impact that each budget receives across the entire set of perturbations.
—*Cross-Budget Heatmap*. These summarize the impact of each perturbation across the entire budget range that the decision maker is interested in.

A crucial component of our tool is the choice of NRP solver to plug in. In this article, we report the results of the use of an implementation of the Nemhauser–Ullmann's algorithm, using this version of OATSAC to experimentally assess the scalability of this algorithm for solving multiple NRP problems as a component to RSA. We also replace this component with a faster, but less precise, greedy algorithm, showing how this is unsuited to the RSA problem, though it is faster, just because it is not exact.

3.1. NRP Solver Module: Nemhauser–Ullmann

The NRP Solver must repeatedly execute NRP instances, each of which has undergone some perturbation with respect to the original one to simulate estimation errors. Since there are many such executions required, and obtaining exact solutions is NP-hard, a great deal of care is required in the design and implementation of this module.

One important family of pseudopolynomial algorithms [Garey and Johnson 1978, 1979] that can be applied to the NRP can be obtained by dynamic programming. In particular, *dynamic programming by costs* is an efficient form of implementing the corresponding Bellman's equation when the number of requirements and the budget are moderate.

$$z(S,B) = \begin{cases} 0 & \text{if } S = \emptyset \\ z(S - \{x\}, B) & \text{if } S \neq \emptyset \wedge cost(x) > B \\ \max\{z(S - \{x\}, B), z(S - \{x\}, B - cost(x)) + revenue(x)\} & \text{otherwise.} \end{cases} \quad (2)$$

In Eq. (2), the set S represents a finite collection of requirements, B is the budget, z is the maximum revenue that can be achieved with any $R \subseteq S$ within the budget constraints, and x is an arbitrary requirement in S whose cost is $cost(x)$ and its revenue is $revenue(x)$.

However, efficient implementations of this scheme typically require integer costs and thus they are not appropriate for RSA, because perturbations naturally produce fractional values. Of course, both the costs and the budget can be scaled to avoid fractional values while retaining precision, but then a previously modest size problem may be transformed into an unfeasibly demanding problem. Whether this unfeasibility occurs in practice will depend upon the specific instances considered. If there is a fine-grained distinction between true costs, then scaling to ensure integer revenues will result in a (potentially) exponential explosion in problem size. Since a problem can have an arbitrary precision of revenues for requirements scores and costs (so arbitrarily fine grained revenues), we cannot be sure that this exponential increase will not occur in practice.

Therefore, though such a scaling approach might be applied in some instances, it is unlikely to provide a general solution in a reasonably large set of cases. The decision maker is unlikely to be an optimization expert and aware of the intricacies of the algorithm. Were such a non-specialist decision maker to rely on a technique that used scaling, we could never be sure that the algorithm at the heart of the approach would not (unexpectedly and inexplicably from the decision maker's point of view) fail to give

an answer in reasonable time. For this reason we need to turn to an alternative way of providing exact scalable solutions to multiple instances of the NRP.

Approximation algorithms are far more efficient than exact algorithms, typically polynomial versus exponential. However, they might mislead the decision maker who is confronted by a sensitivity analysis involving hundreds of solutions corresponding to perturbations of a given NRP instance. Deviations from exact solutions may highlight the wrong requirements as being responsible of changes in the total revenue resulting in ill-informed decisions and sensitivity analysis that is simply wrong.

Therefore, we introduce Nemhauser–Ullmann's algorithm (NU) at the core of the NRP Solver component of OATSAC. We also implemented a greedy approach for comparison and present results (in Section 6.4) that demonstrate that its inaccuracies are problematic, as predicted, thereby motivating our proposed approach, based on an exact algorithm.

ALGORITHM 2: Nemhauser–Ullmann algorithm for NRP

Input: A set of n requirements, R, and a budget, B
Output: A set of selected requirements, S

```
(1)   ▶ Compute the optimal values
(2)   m[0] ← zero()
(3)   for k ← 1 to n
(4)       f ← translation(m[k − 1], cost(R_k), revenue(R_k))
(5)       m[k] ← maximum(m[k − 1], f)
(6)   ▶ Recover an optimal solution
(7)   S ← Ø
(8)   for k ← n to 1
(9)       if apply(m[k], B) ≠ apply(m[k − 1], B)
(10)          S ← S ∪ {R_k}
(11)          B ← B − cost(R_k)
(12)  return S
```

Our particular rendition of NU for NRP is presented in Algorithm 2. The algorithm was implemented in standard C++. Our description is based on the notion of *staircase function*. A staircase function is just an increasing function with a finite number of steps. These functions can be readily represented as lists of pairs containing the coordinates of each step.

First, the algorithm computes an array m of $n + 1$ staircase functions. The function $m[k]$ represents the optimal values that can be obtained by taking into account just the first k requirements.[2] When a new requirement is considered, the previous function is translated by adding the corresponding cost and revenue to each of its pairs. The resulting staircase function, f, may be better than the previous one at some points, while being worse at others. The maximum of both functions represents the combined optimal values.

Applying function $m[n]$ to cost B, which is the budget, we get the best revenue that can be obtained when all the requirements are taken into account. A subsequent simple backward search allows recovering the precise requirements involved in an optimal solution. A set, S, containing the indexes of those requirements is returned.

[2]Function $m[0]$ is just the zero function, a convenient special value.

The technically challenging part of the algorithm is the efficient implementation of function *maximum*. This can be done in linear time with a method resembling the merge stage of mergesort. Observe that *maximum* is always invoked on two functions with the same number of points, say p_{k-1} for $k \geq 1$ with $p_0 = 1$. Nevertheless, the resulting function can have up to $p_k = 2p_{k-1}$ points. Therefore, it follows that in the worst-case situation where such a pathological case would happen again and again, the algorithm would take an exponential amount of time in n.

3.2. NRP Solver Module: Greedy Algorithm

In order to investigate the benefits of using an exact algorithm, our case studies (Section 6) also report results for a greedy algorithm. Given a set of all requirements, R, and a budget, B, the greedy heuristic used in the study selects requirements in decreasing order of the revenue per cost ratio. When tied, requirements with higher revenue get prioritized; ties in revenue are, in turn, resolved in favor of the requirement with lower cost. The pseudocode of this greedy algorithm is presented as Algorithm 3.

ALGORITHM 3: Outline of Greedy Approach for NRP

Input: A set of n requirements, R, and a budget, B
Output: A set of selected requirements, S

```
(1)   ▶ sort R with descending order of revenue(R_i)/cost(R_i)
(2)   sort(R)
(3)   ▶ Greedily select the most promising requirements in budget
(4)   S ← Ø
(5)   k ← 1
(6)   while k ≤ n ∧ B ≠ 0
(7)       if cost(R_k) ≤ B
(8)           S ← {R_k} ⋃ S
(9)           B ← B − cost(R_k)
(10)      k ← k + 1
(11)  return S
```

4. RESEARCH QUESTIONS

In order to evaluate whether our approach and the OATSAC tool that implements it are efficient and useful for the RSA problem, we conduct an experimental study concerning the efficiency of the approach and a set of case studies to explore its usefulness as a tool for revealing otherwise undiscovered sensitivity in budget and requirements. Finally, we compare the results obtained from the exact approach with those obtained from the faster, less precise approach, based on the greedy algorithm to explore the potential this might have to mislead the decision maker.

This is formalized as three research questions for which the rest of the article presents results. First, the need to use exact algorithms for sensitivity analysis raises an important question: what is the realistic cost of using an exact algorithm for sensitivity analysis of NRP? This question is, in turn, formulated more precisely as the following research questions.

RQ1. Impact Factors. Which factors affect the execution time of the exact algorithm for NRP?

RQ2. Scalability Measurement. Based on *RQ1*, how well does the exact algorithm for NRP scale?

In addition to the experimental study of scalability, the article also presents the results of a case study that considers realistic scenarios in which our approach to sensitivity analysis can help the decision maker. The case study uses a real-world requirement dataset from Motorola and investigates the insights that can be provided to the decision maker that would not be possible without the aid of the sensitivity analysis.

RQ3. Insight. What are the possible benefits that a decision maker can expect from using the sensitivity analysis?

RQ1 and *RQ2* are answered by statistically analyzing the measured execution time of the exact algorithm. *RQ3* naturally requires a more qualitative approach, based on usage scenarios for real-world NRP data.

5. SCALABILITY STUDY

The first two research questions are studied by generating synthetic problem instances of different sizes on which we apply the Nemhauser–Ullmann algorithm. The synthetic problem instances are not only varied on their sizes, but also on their difficulty, which will be explained in the following.

5.1. Generating Problem Instances

The research questions outlined in Section 4 require two control variables: the size of the problem (i.e., the number of requirements) and the difficulty of the problem instance. The scalability experiment uses synthetically generated problem instances that correspond to a predefined set of control variables.

While it is trivial to generate synthetic problem instances with predetermined problem sizes, generating problem instances with varying problem difficulties is not, in general. Fortunately, it is known that KP instances become harder when there is higher correlation between the cost and the revenue of items [Pisinger 2005]. Therefore, our synthetic instance generator controls the correlation factor between the cost and the revenue of the requirements in the problem instance in order to control its relative difficulty. Unlike Pisinger, we use Pearson's correlation as the measure of correlation and, thereby, this aspect of problem difficulty. We use Pearson's correlation because it is the most widely used and standard correlation measure used in statistical analysis, whereas Pisinger's correlation is more concerned with easing instance generation and theoretical analysis.

5.2. Experimental Environment

The scalability experiments were performed using a machine with an Intel Core i7 2.67 GHz CPU and 12GB RAM running Ubuntu 10.10. While the CPU provides a multicore environment, the algorithm used in the article has been implemented to run in a single thread. The execution time for each run of the algorithm has been measured using the standard Unix utility, `time`.

In order to answer *RQ1* and *RQ2*, multiple instances of synthetic NRP instances were generated with varying numbers of requirements and relative problem difficulty levels, that is, the correlation between the cost and the revenue. In total, 15 different problem sizes ranging from 100 to 1500 requirements with steps of 100 were used; similarly, the correlation factor ranged from 0% (random costs and revenues) to 100% (equal cost and revenues) with steps of 5%. This results in $15 \cdot 21 = 315$ problem configurations. In order to capture a sufficiently large sample of possible problem

instances for each of the given control parameters, 50 different problem instances were generated from each of the 315 problem configurations. Each instance consists of a set of requirements with costs and revenues that are synthetically generated to uniformly sample the space of possible problem instances with respect to a fixed correlation. For each problem instance, the budget for NRP was set to the half of the total cost of all its requirements. In total, $315 \cdot 50 = 157{,}500$ different instances of NRP were solved using our implementation of the NU algorithm to provide results on scalability.

5.3. Results and Analysis

The results of our analysis of scalability are depicted in Figure 2. Figure 2 (upper graph) shows the box plots of the execution time of NU for NRP against problem instances with increasing number of requirements. Each datapoint corresponds to one execution of the algorithm on a synthetically generated problem instance. The box plots depict the distribution of 50 random problem instances generated for a single problem configuration.

The growing problem size increases the execution time of the algorithm almost polynomially, but the algorithm takes less than 1 minute for a problem instance with as many as 1,500 different requirements. In realistic scenarios, the number of requirements considered for a single release may be significantly fewer than 1,500, which makes our version of the Nemhauser–Ullmann algorithm scalable for multiple executions of NRP for use in an OAT approach to RSA.

Figure 2 (lower graph) shows the box plots of the execution time of Nemhauser–Ullmann algorithm for NRP against problem instances with increasing correlation between the cost and the revenue of each requirement. As with the upper graph of Figure 2, each datapoint corresponds to a single execution of the algorithm. While the increasing correlation factor does result in increasing execution time, the impact is less than that of the increasing number of requirements and the growth is almost polynomial. The only exception to polynomial growth occurs when correlation is close to 1.0. However, the resulting execution time remains feasible, even in such extreme cases.

Table I presents the results for the average time data for configurations (i.e., a pair of correlation and problem size) with 10% correlation factor intervals. Given the number of requirements, n, and the correlation between the cost and the revenue of each requirement, ρ ($0 \leq \rho \leq 1$), the following model of the time behavior, $t(n, \rho)$, fits the experimental data very well:

$$t(n, \rho) = an^2 \exp \rho + bn^2 + cn \log n. \tag{3}$$

Coefficients for Eq. (3) are $a = 6.56 \cdot 10^{-1}$, $b = 4.67 \cdot 10^{-6}$, and $c = -1.14 \cdot 10^{-3}$. Figure 3 shows the plot of the experimental model of the time behavior together with its residuals. The R^2 value for the fit is 0.99. We do not claim that the model explains the behavior of the algorithm under every circumstance, which would be clearly misleading considering that the worst-case execution time for NU is known to be $O(2^n)$ [Nemhauser and Ullmann 1969]. In pathological cases, our approach still may not apply. Nevertheless, as these experimental results indicate, there is strong evidence to indicate that our use of NU scales well to RSA.

6. CASE STUDY

The previous section provided evidence that our overall approach to RSA can scale to handle requirements problems with many requirements and with cost-revenue correlations that are known to cause exponential behavior in the underlying optimization algorithm. However, these results say nothing about whether the overall approach,

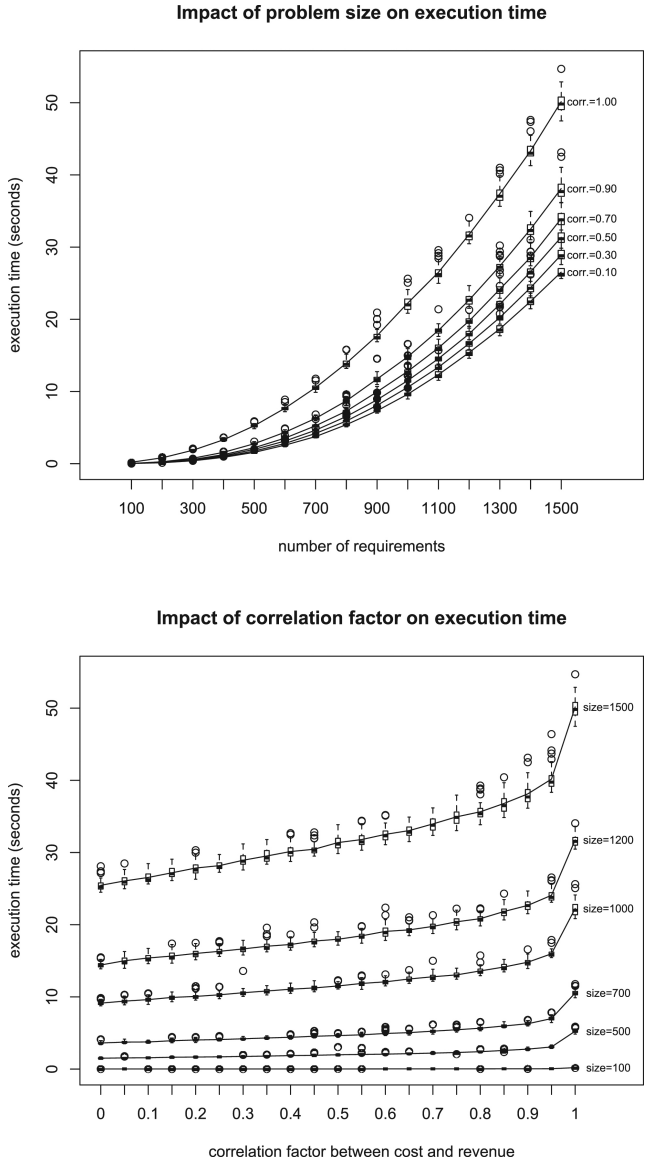


Fig. 2. The upper graph shows plots of execution time (seconds) for our implementation of the Nemhauser–Ullmann algorithm against NRP instances with increasing number of requirements. The lower graph show plots of execution time (seconds) against NRP instances with increasing correlation between the cost and revenue of each requirement. It can be seen from the upper graph that execution time grows almost polynomially. From the lower graph, it can be seen that the impact of higher correlation values is also almost polynomial throughout the range of correlation values (except when the correlation value is very close to 1.0).

implemented in OATSAC, can help find interesting, important or insightful instances of sensitivity in real-world RSA problems.

To address this question, we present a detailed investigation of a requirements analysis problem from Motorola. In this problem, the requirements are potential features for mobile (cell) phones. While the specific details of the features and phones cannot be revealed for confidentiality reasons, this does not prevent us looking for interesting

Table I. Execution Time (seconds) of Nemhauser–Ullmann's Algorithm for NRP for Different Problem Configurations

	Correlation										
Size	0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %	100 %
100	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.04	0.20
200	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.22	0.26	0.81
300	0.38	0.40	0.42	0.45	0.47	0.50	0.53	0.58	0.65	0.76	1.86
400	0.83	0.88	0.91	0.97	1.03	1.09	1.18	1.25	1.35	1.57	3.31
500	1.50	1.57	1.67	1.75	1.86	1.98	2.07	2.21	2.41	2.76	5.25
600	2.43	2.56	2.66	2.80	2.99	3.13	3.29	3.54	3.82	4.35	7.68
700	3.62	3.76	4.03	4.20	4.40	4.64	4.93	5.26	5.66	6.28	10.55
800	5.07	5.40	5.65	5.91	6.32	6.54	6.90	7.28	7.81	8.71	13.92
900	6.99	7.33	7.70	8.07	8.46	8.84	9.31	9.95	10.56	11.73	17.73
1000	9.13	9.62	10.07	10.58	11.06	11.51	12.08	12.79	13.60	14.80	22.25
1100	11.63	12.33	12.88	13.36	14.02	14.55	15.53	16.04	16.94	18.46	26.51
1200	14.42	15.39	16.02	16.64	17.26	17.98	19.07	19.76	20.84	22.67	31.75
1300	17.82	18.67	19.68	20.25	21.21	22.08	23.07	24.17	25.57	27.37	37.39
1400	21.48	22.48	23.49	24.37	25.46	26.57	27.58	28.58	30.32	32.48	43.32
1500	25.46	26.55	27.85	28.93	30.14	31.38	32.51	33.96	35.64	38.12	50.16

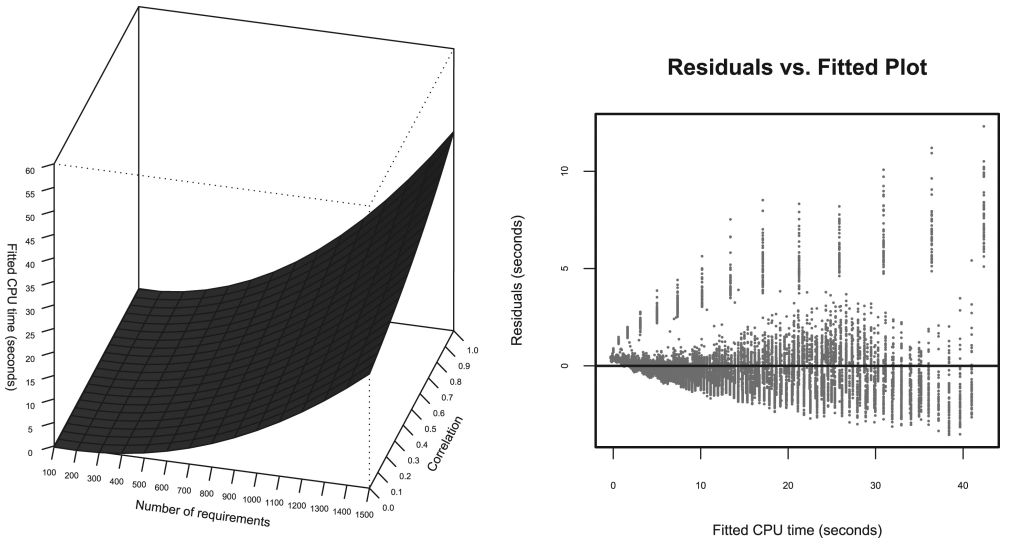


Fig. 3. The plot of the fit from the regression analysis, $t(n, \rho) = an^2 \exp \rho + bn^2 + cn \log n$ and the plot of residuals. The R^2 of the fit is very high (0.99), indicating a very good fit to the observed experimental data.

and potentially insightful instances of peculiarly sensitive budgets and requirements and the interplay between them.

Using this dataset we do, indeed, find instances of both peculiarly sensitive requirements and peculiarly sensitive budget levels. Identifying and investigating these sensitivities would be very hard, if not impossible, without some form of decision support, such as that provided by our RSA approach, implemented in OATSAC.

6.1. Requirements Data

The case study considers a dataset from Motorola that contains 35 independent requirements. The dataset contains the cost of implementation and the expected revenue

for each requirement, which form the cost function *cost* and the desirability function *revenue* in Section 2, respectively. The expected revenue data have been provided by the customers, based on the desirability of having certain features in the next release. The dataset also contains 35 different budget values that are set by human experts.

6.2. Sensitivity Analysis

The sensitivity analysis we performed simulates misestimation of the cost of implementing a requirement by applying different PIAC values to the original cost of a requirement. For example, a positive PIAC value p means the actual cost has increased by p% of the estimation (underestimation). Similarly, a negative PIAC value -p means the actual cost has decreased by p% of the estimation (overestimation).

The sensitivity analysis uses PIAC values ranging from −50% to 50% with 5 percent-point intervals for each requirement. This is a matter of taste and can be varied without changing our approach. We find that this is a good balance of range and granularity. For n requirements, this results in $21n$ different 'what-if' scenarios that need to be solved for the sensitivity analysis, just for a single given budget proposal.

The total impact is computed separately for all positive PIAC values and for all negative PIAC values. The impact for each PIAC value is computed as the difference in overall revenue with respect to the solution of the unperturbed instance. The total impact of a misestimated requirement for each set of PIAC values is the sum of their impacts.

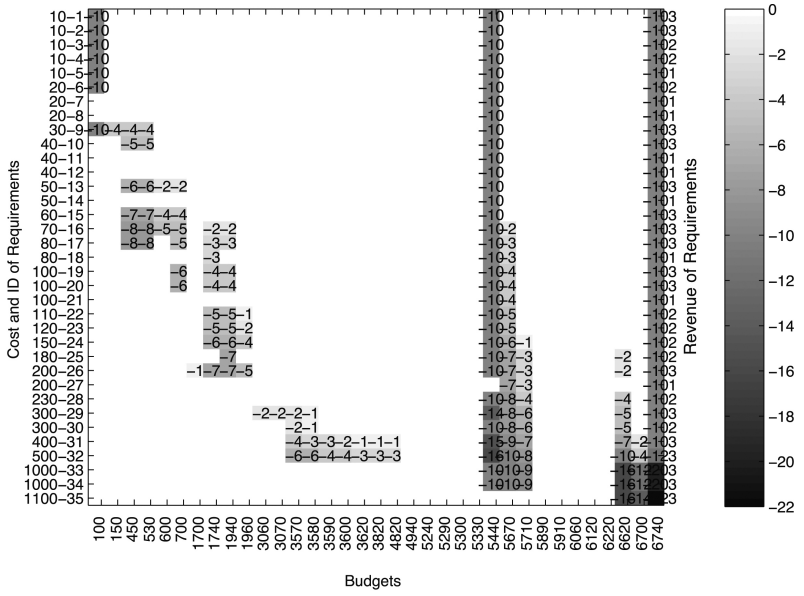
6.3. Results and Insights

In this section, we present the results of two case studies that illustrate how our approach can be used to support decision makers as they consider the various options available in negotiations over budgets. We consider two top-level scenarios for which sensitivity analysis can be useful in practice, applying both to the Motorola dataset.

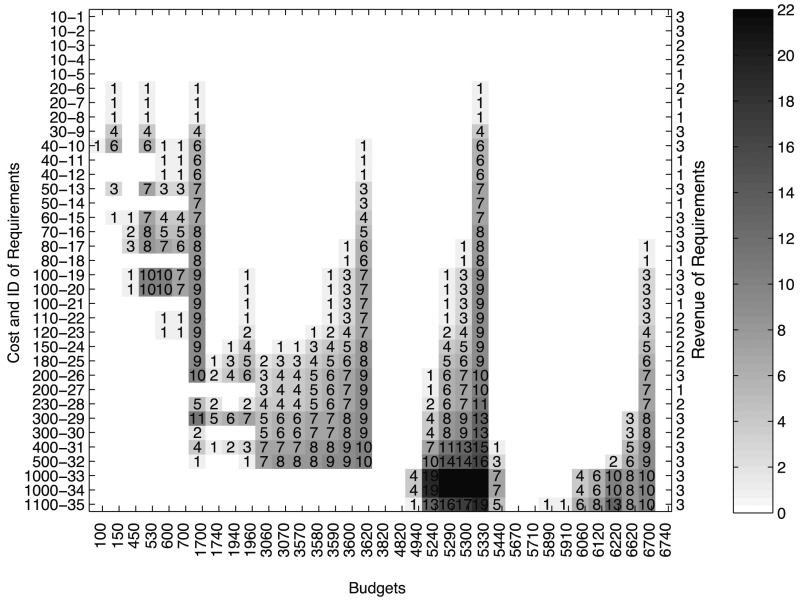
In the first study, we consider a situation in which a budget level turns out to be particularly sensitive over all possible requirement inaccuracies. In the second study, we consider a scenario in which a particular requirement turns out to be atypically sensitive, regardless of the budget chosen. In this way, these two studies illustrate the two top-level concerns for the decision maker: sensitive budgets and sensitive requirements.

We will use the OATSAC heat map visualization of the sensitivity data in order to locate sensitive budgets and requirements. As the results show, the use of heat maps allows the decision maker to quickly and easily identify unusual areas of sensitivity within their candidate solution space. The scenarios show that sensitive budgets and requirements can easily be identified using the sensitivity heat map. However, they also reveal that the explanation for a particular sensitivity can be nontrivial and thereby non-obvious without the aid of OATSAC.

6.3.1. Sensitive Budgets. Consider the heat maps in Figure 4. These two heat maps show a perspective of our RSA results for the Motorola data set. The total impact of positive PIACs (Figure 4(a)) and negative PIACs (Figure 4(b)) on each budget (X-axis) for every requirement (Y-axis) is illustrated by the degree of darkness on the heat maps. Zero impact is represented as white color on both heat maps. Although darkness represents the difference on revenue caused by misestimation, in Figure 4(a), it represents the loss on revenue caused by +PIACs on cost of requirements, whilst in Figure 4(b), it represents the gain on revenue caused by −PIACs on cost of requirements. For example, consider requirement 13 for budget 450: the sum of all losses in revenue for the positive PIAC values is 6.



(a) Total impacts of +PIACs. Vertical bars indicate budgets {5440, 6740} are sensitive to +PIACs.



(b) Total impacts of -PIACs. Vertical bars indicate budgets {1700, 3620, 5330} are sensitive to -PIACs.

Fig. 4. Budgets {5440, 6740} have zero surplus budget, which makes them sensitive to underestimates (+PIAC). On the other hand, budgets {1700, 3620, 5330} have relatively more surplus budget {5.3%, 4.9%, 13%}; as a result, they are sensitive to overestimates.

Some allocations do not fully use the available budget creating a small budget surplus. This happens, for a solution s, when the total cost of solution s is lower than the available budget, but no other solution exists that is within budget and enjoys a higher value. In such situations, there is a budget surplus (computed as the amount of budget available minus the total cost of the requirement selected for solution s). As shown in Figure 4, in general, budgets with surplus close to zero are more sensitive to +PIAC, and budgets with "rich" surplus are more sensitive to −PIACs. For example, the increasingly dark area on the left to the budget 5,330 in Figure 4(b) and the decreasingly dark area on the right to the budget 5,440 in Figure 4(a) are both caused by the increasing percentage of surplus budgets from left to right around that area.

For the heat map in Figure 4(a), starting from budget 5,440 to the two budgets on its right, because the surplus is growing away from the level of zero, it becomes easier to maintain the same level of overall revenue when the cost of the selected requirement was underestimated (+PIAC). This is reflected on the heat map as a trend of decreasing impact on the right of budget 5,440, meaning that budgets' ability of absorbing error is increasingly stronger when surplus is growing.

On the other hand, considering the four budgets on the left of budget 5,330 for the heat map in Figure 4(b), because the surplus is increasing to "richer" levels from left to the right, it becomes easier to accommodate the extra unselected requirements when selected requirement was overestimated (−PIAC). This is reflected on the heat map as the trend of increasing impact on the left of budget 5330, meaning that increasingly more revenue is added into solutions when surplus is growing.

More specifically, let us observe the dramatic sensitivity at the following budgets: $\{1{,}700, 3{,}620, 5{,}330, 5{,}440, 6{,}740\}$. Their sensitivity is revealed by the noticeably darker colors present around these budget levels. Closer analysis of the data reveals that these sensitive budgets can be classified into two types depending on the level of its surplus budget.

—*Type 1*. In budget 5,440 and 6,740, as shown in Figure 4(a), the original optimal requirement assignment has available very little surplus budget.
—*Type 2*. In budget 1,700, 3,620, and 5,330, as shown in Figure 4(b), there is a relatively large surplus available in budget: $\{5.3\%, 4.9\%, 13\%\}$.

Clearly for Type 1 budgets, the lower the budget surplus, the less room for maneuver should there be for an inaccuracy in the estimation of costs. This can be expected to be a general trend. Indeed, for the Motorola data set, there is a good fit for an exponential function $f(x) = 356.0 \exp(-139.8x)$ to the graph of sensitivity to budget surplus, as shown in Figure 5.

One might be tempted to think this applies to all low surplus budgets, declaring these all to be sensitive budgets. However, this approach would be unreliable. For instance, observe that, though Figure 5 reveals a clear overall trend, it also contains notable outliers. This means that one cannot simply assume that low budget surplus leads to high sensitivity. For example, the budgets 100 and 600 have zero surplus, which is the tightest of all budgets for any solution. These turn out to be unremarkable and not particularly sensitive budgets. This is so because it turns out that there are plenty of unselected requirements with similar costs that can be substituted for misestimated requirements.

By contrast, the budget 5,330 is a rather sensitive one, as revealed by the heat map in Figure 4(b), yet it has a relatively high budget surplus. This is due to the fact that high budget surplus makes a budget sensitive to negative PIACs because of decreasing cost of selected requirement can make more room for either (1) accommodating extra unselected requirements, or (2) substitution by requirements combinations with higher revenue. This type of budget was classified as Type 2 sensitive budgets.

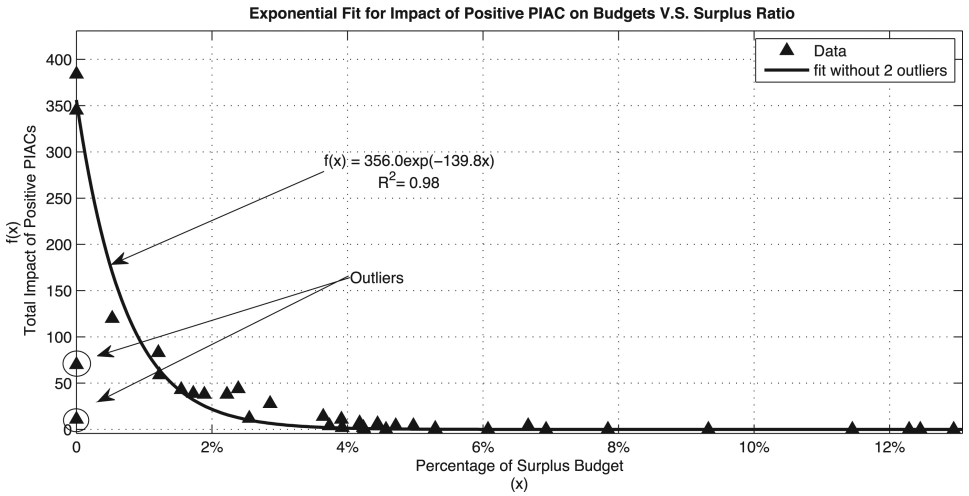


Fig. 5. Strong correlation between impacts of +PIAC and surplus percentage (*Spearman Correlation Coefficient:* $\rho = 0.92$). However, two outliers: budgets 100 (lower) and 600, have zero surplus on budget but they are not sensitive to underestimated cost of requirement (+PIACs) because of the flexibility of substitutions.

Furthermore, one cannot assume, even for a budget identified as sensitive, that all requirements will be sensitive, nor that all levels of inaccuracy will be equally important. For example, consider again, the budget level 5,440, which was revealed to be a sensitive one for Figure 4(a). Observe that requirement 27 is not at all sensitive at this budget level, even though all other requirements are for this budget level. In this case, further detailed analysis reveals that this requirement has the same cost as another requirement, and that this means that the other requirements can be chosen in preference to it, making it unimportant at that particular budget level.

Once again, one might be tempted to adopt the assumption that all such "equal cost" requirements would be similarly insensitive, but once again, this would be a misplaced assumption; requirement 13 has identical cost to requirement 14, yet both are sensitive at this highly sensitive budget level of 5,440, as the heat map shows on Figure 4(a). This visual illustration of sensitivity can be useful to the decision maker. We discuss, in more detail, how a decision maker might find actionable results using our heat maps in Section 8.

6.3.2. Sensitive Requirements. The previous section illustrated how the OATSAC heat map can help to identify sensitive budget levels. In this section, we turn our attention to using OATSAC to identify sensitive requirements.

Consider the OATSAC heat map in Figure 6 which shows the impact of each requirement by each level of PIAC. The level of darkness on the heat map represents the degree of difference caused by corresponding PIAC (horizontal axis) on a specific requirement (vertical axis) summarized over all budgets. The corresponding requirements on the vertical axis are arranged in ascending ordered of their cost, and 'equal cost' requirements are arranged in descending order of their revenue.

In general, requirements with higher costs have higher sensitivity to errors, because applying the same percentage of error (PIAC) on the cost of requirements, the most expensive requirements tend to entail a larger absolute error. This can be expected to be a general trend. Indeed, as shown in Figure 7, the level of sensitivity of a requirement has a strong correlation with its cost. The statistical analysis also confirms this claim with a value of $\rho = 0.94$ for the Spearman Correlation Coefficient. Requirement

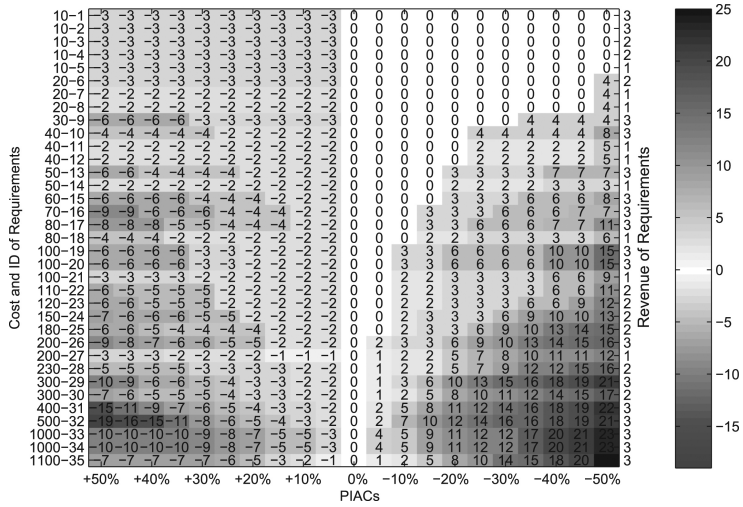


Fig. 6. The impact of each requirement with different levels of misestimation.

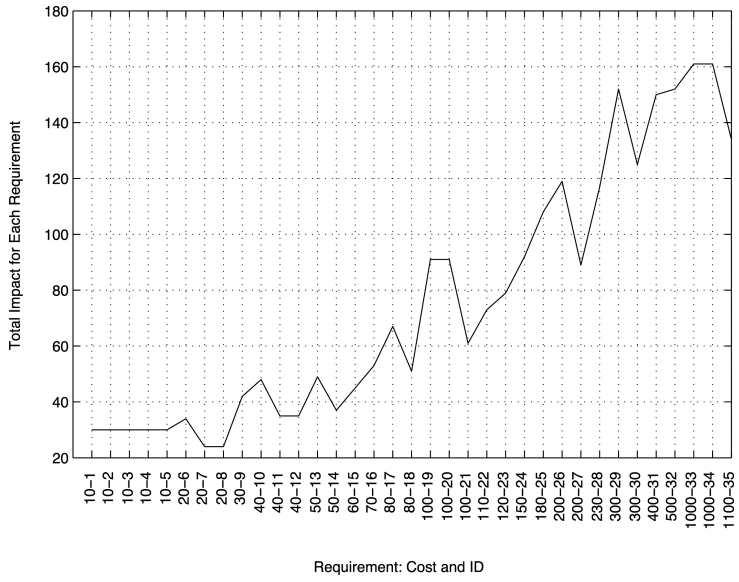


Fig. 7. Cost of requirements vs. total impact of each requirement. More expensive requirements (on the right) do not necessarily have more impact than the cheaper (i.e., to the left) ones. When the costs are the same, those with lower revenue (on the right) have lower impact, for example, these requirement sets have identical cost: {6, 7, 8}, {10, 11, 12}, {13, 14}, {17, 18}, {19, 20, 21}, {26, 27}, and {29, 30}.

revenues also play an important role during the selection process. Naturally, one would expect that when two requirements have the same cost but different revenue, the requirement with lower revenue is always less sensitive.

However, though these are general expectations, there are exceptions, and the OATSAC heat map helps to reveal these exceptions. Requirement 35 has a higher cost and with the same revenue as requirement 34, but, as revealed by the heat map in Figure 6, requirement 35 has a relatively smaller impact than requirement 34.

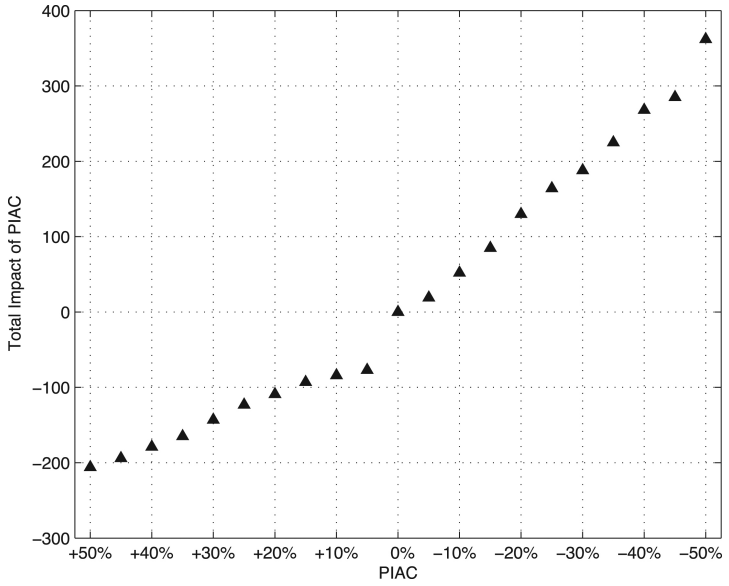


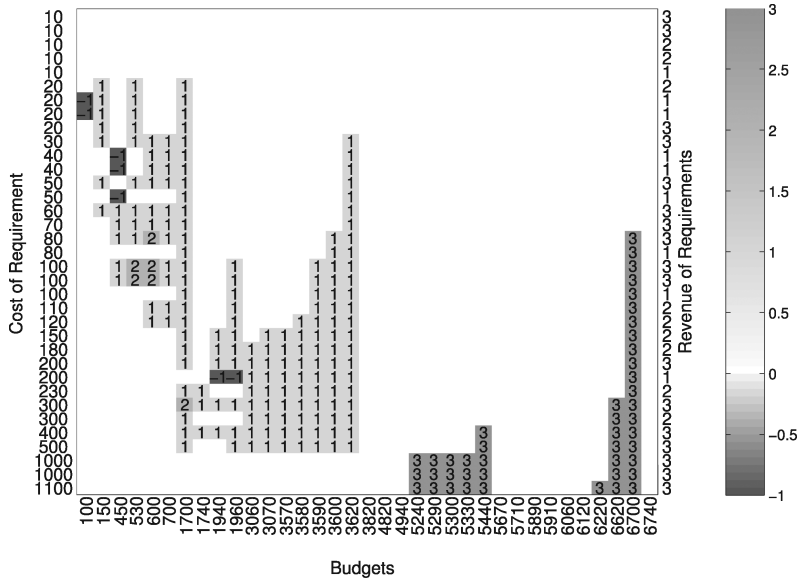
Fig. 8. Rank correlation between the total impact of each PIAC and PIAC value. The trend is monotonic.

6.3.3. Sensitivity to Specific Levels of Misestimation (PIAC). There is another interesting observation, as shown in Figure 8: underestimations on the cost of requirements (negative PIAC, shown on the right half of the figure) have higher absolute impact than overestimations. Closer examination of the data reveals that this is due to the large number of available substitute requirements in the Motorola dataset. That is, it turns out that often there are two or more equally good solutions with same revenue in this particular dataset so that the misestimation of one requirement value might not affect the overall solution; the solution can be retained by a like-for-like substitution. By contrast, when the PIAC is negative, indicating an underestimation of requirement cost, either more surplus budget becomes available, or the cost of some unselected requirement becomes sufficiently small that it can be accommodated within the increased surplus.

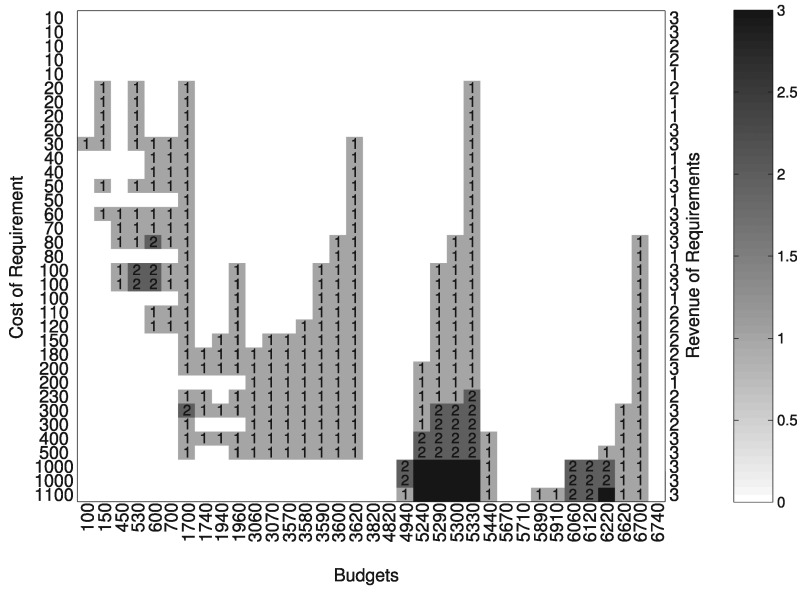
6.4. Why the Use of an Exact Algorithm is Essential for RSA

The greedy algorithm provides a nonstochastic approximation for NRP. Although the deterministic nature of the algorithm makes it an ideal candidate for an OAT approach, the errors in approximated solutions can mislead the decision maker. In this article, we demonstrate the negative impact of using a non-exact heuristic for SA. The greedy algorithm used in this comparison selects requirements in the descending order of their revenue-to-cost ratio, subject to the constraint that the total cost does not exceed the given budget.

Figure 9 illustrates the comparisons of the results for NU-OATSAC with Greedy-OATSAC for a PIAC $= -50\%$. While Greedy-OATSAC produces results that are superficially similar to NU-OATSAC, there are pairs of requirements and budgets that show very different levels of impact. Certain impacts are wrongly either ignored or exaggerated by Greedy-OATSAC. For example, the darker areas in the Greedy-OATSAC results represent lost revenues, which would be regarded as a significant impact. However, the NU-OATSAC shows that the revenue can be retained. This illustrates the importance of using more expensive (but exact) algorithms for RSA.



(a) Greedy algorithm.



(b) Nemhauser–Ullmann's algorithm.

Fig. 9. Comparison of greedy and NU NRP Solver results. The two graphs show the impact on total revenue with 50% overestimation (PIAC = −50%) on the cost of each requirement according to each algorithm. Note that the greedy algorithm misreports many heat map values. This can mislead the decision marker. The NU algorithm is exact and so all reported heat map values are guaranteed to reflect only the impact of sensitivity on the NRP, while for greedy, the heat map results are confounded by the inherent imprecision of the algorithm.

7. CATERING FOR HIGHER ORDER ESTIMATE INACCURACY EFFECTS

In this section, we consider the problem of interactions between estimate inaccuracies. Clearly, as the number of interactions between estimate inaccuracies increases, the computational cost of accounting for them grows exponentially. We call the problem of analyzing interactions between estimate inaccuracies involving two different requirements the *second-order interaction problem*. More generally, we use the term *higher-order interaction* for any interaction between inaccuracies involving n requirements for $n > 1$.

In order to cater for second-order interaction problem, we need a different approach to visualization. However, at this order, it remains possible to consider all possible interactions, giving the decision maker a complete picture of the possible impact of estimate inaccuracies, in which two inaccuracies occur simultaneously. The problem of analysis of higher orders remains a topic for future work (discussed briefly in Section 12).

We use a four-dimensional format to display the second-order interactions for a given level of estimate inaccuracy (PIAC). We use a three-dimensional plot for the axes relating to the two requirements to which the PIAC is applied and show the corresponding budget as the third axis. In order to capture the impact of the interaction between the two requirements for a given budget and PIAC level, we show this as a color intensity (the 'fourth' dimension). Naturally, this style of visualization is best viewed and explored interactively in color, though it can also be appreciated in black and white, through the corresponding shading.

In order to assess the impact of second-order effects, we measure the additional impact obtained from the interaction of the two requirements over and above the impact observed for the sum of each of the individual impacts. This provides a visualization of the additional effects due the interaction of estimate inaccuracies (the second-order effects). Note that the additional impact can be positive or negative, as it is the difference between the joint impact of two requirements and the sum of their individual impacts, and the former may be larger or smaller than the latter.

Figure 10 shows the effects of underestimates, when the degree of underestimation is the maximum considered in this article. That is, in this figure, the true cost is 50% greater than estimated, so PIAC $= +0.50$). Figure 11 shows the effects of overestimates, when the degree of underestimation is the maximum considered in this article.

As can be seen from these two figures, the additional impacts that accrue from second-order effects tend to impact most on the more expensive budgets (the budget levels are ordered in increasing size of the vertical axis). The two figures also reveal that there are certain budget levels that suffer more than others from sensitivity to second-order estimate inaccuracies. Looking at these budgets we observed that there was a great deal of similarity between those budgets that are sensitive at the first order and those that are sensitive at the second order.

This may provide tentative evidence that the additional effects that accrue from higher-order effects are closely coupled to their first-order counterparts; budgets that are sensitive at the first-order level tend to be sensitive at higher orders. Of course, more examples need to be considered, and more analysis of higher order effects would be required to provide sufficient empirical evidence to support any generalization of this claim. It may also be possible to demonstrate a theoretical relationship between first-order effects and additional impacts at higher orders. However, this remains a problem for future work.

The effects of second-order interactions can also be seen for lower estimate inaccuracy levels (smaller absolute values of PIAC). These are shown in Figures 12 and 13 which show, respectively, additional second-order effects of under- and overestimation for four other PIAC levels. Naturally, the lower the estimate inaccuracy, the lower the impact of any and all estimate inaccuracies. However, we see the same overall pattern in these

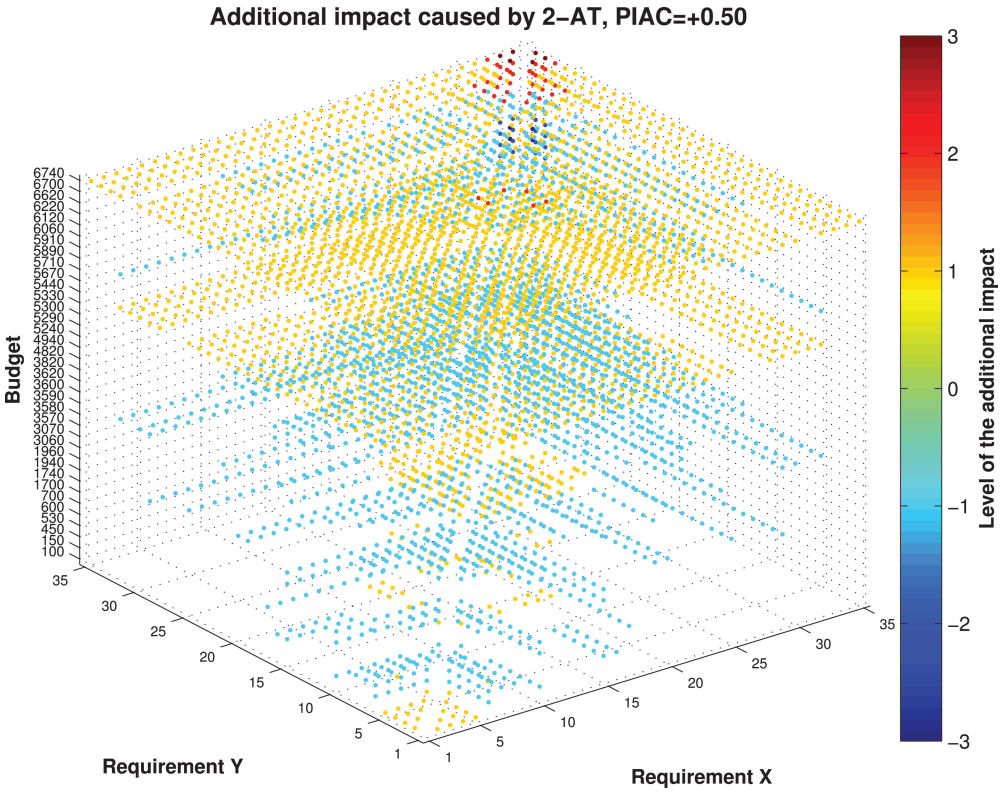


Fig. 10. Additional impact of second-order effects for 50% underestimated cost (PIAC = +0.50).

figures: certain budgets are more sensitive than others, and there is a general tendency for more expensive budgets to suffer from greater additional impact.

8. HOW THE RESULTS CAN BE USED BY A SOFTWARE ENGINEER

We saw (Figure 4) that there is a general principle empirically observed in the Motorola dataset that budgets with surplus close to zero are more sensitive to underestimation, while budgets with 'rich' surplus are more sensitive to overestimates. However, we also saw that there are sensitivities that can only be revealed by the analysis and do not follow this general principle. For example, the budget values 100 and 600 have zero surplus (tightest of all budgets) but are not particularly sensitive, while budget 5,330 is highly sensitive considering it has a relatively high budget surplus.

The decision maker can thus use the heat maps as a way to identify those budgets that are sensitive to estimate inaccuracies. This can feed into the negotiations the decision maker might have with other stake holders. Without the heat maps, the decision maker would simply have to assume that tight budgets would be sensitive and that high surplus provided a cushion of security. As we have seen, this would not be the best policy. There may be more room to maneuver in a tight budget scenario and any sense of security arising from a budget surplus might be a very false sense of security. The decision maker would thus be well advised to use the analysis afforded by the heat maps to back up their innate common sense and intuition.

Similarly, we observed that the degree of sensitivity of requirements varies (even within budget levels that are, themselves, found to be very sensitive). For example, the

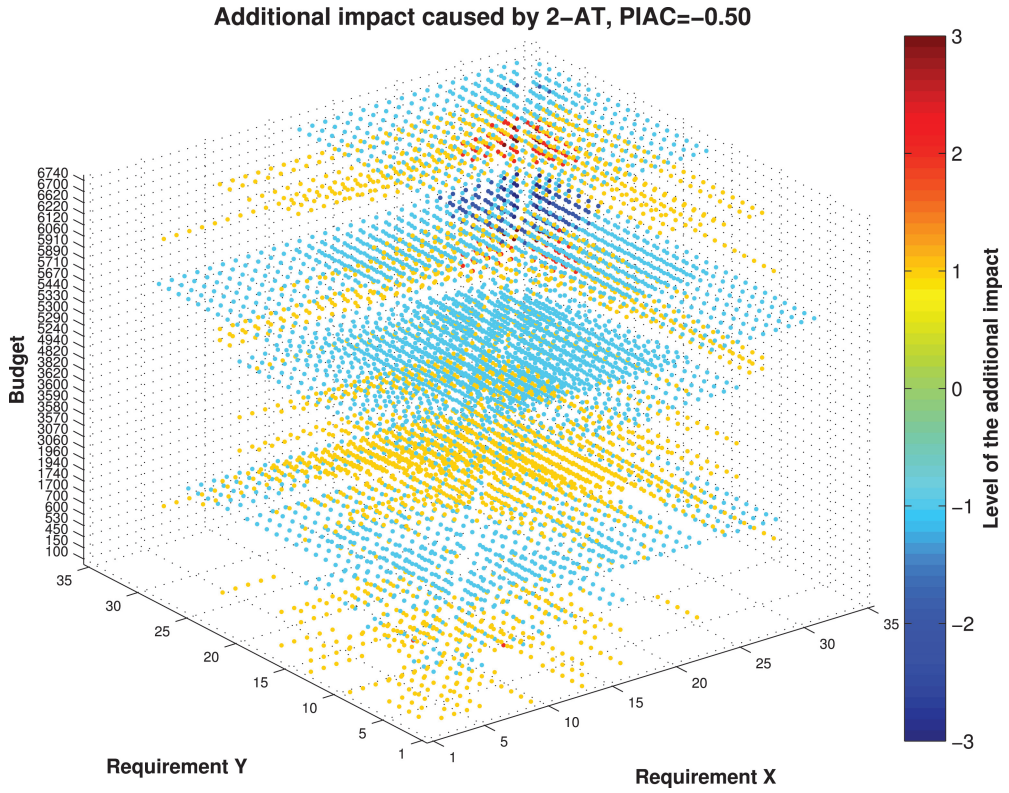


Fig. 11. Additional impact of second-order effects for 50% overestimated cost (PIAC $= -0.50$).

budget 5,440 illustrates this point very clearly. Even for this highly sensitive budget level, there is one completely insensitive requirement that remains entirely unaffected by estimate inaccuracy.

These kinds of observations, taken directly from the heat maps, also provide information that can be useful to the decision maker. The analysis of the heatmaps reveals that it would be wise to seek to negotiate for a different budget if presented with a management case for an overall budget of 5,440. Furthermore, it supports the decision maker by providing him or her with a business case to underpin their negotiations. The decision maker might appeal: "this particular budget is simply a too risky budget and a dramatic reduction in risk can be achieved with a small budget modification."

Of course, such negotiation may prove to be either impossible or unsuccessful. However, even in such an unfortunate situation, our sensitivity analysis approach retains its usefulness, because the decision maker is both alerted to the need for particularly careful cost estimation and also to those requirements that require particular attention.

We have seen that the decision maker can identify highly sensitive budget levels and requirements in the Motorola dataset, providing some evidence that the use of our heat maps can provide actionable results to the software engineering decision maker. We cannot claim that all requirements problems will yield interesting actionable results, but even where they do not, this means that the manager will have additional confidence that 'there is nothing out of the ordinary' in the sensitivity of the budgets and requirements.