Configuring Release Plans

Felfernig, Alexander

CEUR-WS.org
2018-09-23


http://hdl.handle.net/10138/298427

Downloaded from Helda, University of Helsinki institutional repository.
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.
Please cite the original version.
Configuring Release Plans


Abstract. Release planning takes place (1) on the strategic level where the overall goal is to prioritize (high-level) requirements and (2) on the operational level where the major focus is to define more detailed implementation plans, i.e., the assignment of requirements to specific releases and often the assignment of stakeholders to requirements. This paper shows how release planning can be represented as a configuration task and how re-configuration tasks can be supported. Thus, we advance the state-of-the-art in software release planning by introducing technologies that support the handling of inconsistencies in already existing plans.

1 Introduction

Higher flexibility of software development and better satisfied customer requirements can be achieved by developing and delivering software in an incremental fashion [8]. Release planning is needed to support such development approaches in a structured fashion. Release planning can be regarded as a company-wide optimization problem where stakeholders want to maximize the utilization of often limited resources [8, 10, 11]. Planning often takes place on two levels [1]. First, on a strategic level where the major task is to prioritize requirements with regard to criteria such as business relevance (profit), feasibility (risk), and related efforts [9, 11]. On an operational level, a detailed planning takes place where requirements are assigned to releases and often also to stakeholders in charge of their implementation. The consequences of poor release planning are low software quality, lost business opportunities, more replanning efforts, and also project cancellation [10].

Figure 1 depicts an overview of a release planning process. First, requirements are prioritized on the basis of a utility analysis [3]. Second, detailed planning takes place where requirements are assigned to releases and a release planner is in charge of assuring the consistency of the plan with regard to a set of additional constraints related to dependencies between requirements and further constraints imposed by stakeholders (see the example release constraints depicted in Table 5).

The major contributions of this paper are the following. First, we show how to represent a release planning problem as a configuration task. In this context, we also show how re-configuration can be supported on the basis of configuration and diagnosis techniques. Second, we report the results of a performance analysis that has been conducted on the basis of different types of release planning (configuration) problems.

The remainder of this paper is organized as follows. In Section 2, we sketch how utility analysis can be performed on the basis of a given set of requirements. Thereafter, in Section 3 we show how release planning can be represented as a configuration task and how re-configuration can be supported in this context. In Section 4, we report the results of an evaluation of the proposed approach. The paper is concluded with a discussion of issues for future work.

2 Utility-based Prioritization of Requirements

The prioritization of requirements can be performed on the basis of a utility analysis, i.e., the evaluation and ranking of requirements with regard to a predefined set of interest dimensions such as profit, effort, and risk [3]. In the line of the two basic recommendation approaches in group recommender systems [4], prioritization can be performed in two ways (see Figure 1): (1) aggregated utilities based approaches collect stakeholder-individual requirements evaluations with regard to a set of interest dimensions, aggregate those evaluations, and calculate requirement utilities thereof. (2) aggregated prioritizations based approaches aggregate stakeholder-specific prioritizations into the final prioritization. In the following, we explain both approaches in more detail. The second variant assumes that each stakeholder provides a prioritization (directly or in terms of utility evaluations) whereas the first approach also allows prioritization in situations where not all stakeholders evaluated each of the given requirements.

Prioritization with Aggregated Utilities. In this context, multi-attribute utility theory can be applied to determine a ranking (prioritization) of a given set of requirements (see Table 1). First, each stakeholder $s_i$ evaluates each individual requirement with regard to interest dimensions. In our example, we use the interest dimensions profit, effort, and risk, which are typical interest dimensions in release planning. Interest dimensions can have an assigned weight, for example, it is more important to avoid risky release plans than maximizing the potential profit.

In such a setting, the distribution of weights could be similar to the one defined in Table 2. Second, on the basis of a given set of evaluations and a specification of the importance of individual interest dimensions, requirements can be ranked according to Formula 1 where $imp(d)$ denotes the importance of interest dimension $d$ and $contrib(r,d)$ denotes the contrib-
utility(r) = Σ_{d ∈ Dim} imp(d) × contrib(r, d)  \tag{1}

When applying Formula 1 to the entries in Table 1 and Table 2, we are able to derive a ranking of the requirements \( R = \{r_1, r_2, ..., r_n\} \) as depicted in Table 3.

In this paper, we directly evaluate requirements with regard to interest dimensions (on a rating scale [1..10]). Especially for the dimensions effort and profit, alternative evaluation scales can be defined which are then mapped to a utility scale (e.g., [1..10]). A utility analysis can be regarded as test cases for the prioritization process (see Definition 1).

Definition 1: Consistent Prioritization: given a prioritization \( P = \{p_1 : r_1 = v_1, p_2 : r_2 = v_2, ..., p_n : r_n = v_n\} \) for requirements \( REQ = \{r_1, r_2, ..., r_n\} \) \( (v_i ∈ \text{domain}(r_i)) \) and a set of test cases \( T = \{t_1, t_2, ..., t_k\} \). Then \( P \) is a consistent prioritization if \( P ∪ t_i \) is consistent \( ∀t_i ∈ T \).

Consistent prioritizations determined on the basis of an utility analysis can be regarded as input for release planning. In the following, we show how prioritizations can be exploited in the context of release planning and how release planning can be represented as a constraint-based configuration task.

3 Constraint-based Release Configuration

Constraints \( r_i = v_i \ (v_i ∈ \text{domain}(r_i)) \) representing individual requirement prioritizations can be used as one input of a release configuration problem [5] by generating release assignment constraints following the rule \( ∀\{p_i : r_i = v_i, p_j : r_j = v_j\} ∈ P (i ≠ j) : v_i > v_j → rel_{ij} ≥ rel_{ij} \). Since \( r_1 > r_2 \) holds in our working example, we can derive the constraint \( pc : rel_{ij} ≥ rel_{ij} \) as an input for our release configuration task (see Definition 2). We denote the set \( \cup pc \) derived from a prioritization \( P \) as \( PC \).

Definition 2: Constraint-based Release Configuration Task: a constraint-based release configuration task can be defined as a tuple \( (R, PC, RC) \) where \( R = \{rel_{i1}, rel_{i2}, ..., rel_{in}\} \) is a set of variables representing potential assignments of requirements to releases \( \text{domain}(rel_{i}) = [0..n] − 0 \) refers to requirements not assigned to a release, \( PC = \{pc_1, pc_2, ..., pc_m\} \) represents a set of prioritization constraints \(^5\), and \( RC = \{rc_1, rc_2, ..., rc_l\} \) is a set of release constraints.

Examples of typically used release constraints are given in Table 5. Further release constraints that will not be taken into account in our working example are release capacity in hours, total capacity in hours, release costs, total costs, assignment of stakeholders to requirements, average risk level per release, minimum market value per release, and further optimization.

\(^4\) Hard prioritization constraints are often too strict in practice. They can also be interpreted as preferences, i.e., solvers should take these into account as much as possible.

\(^5\) For an overview of different types of preference aggregation functions we refer to [4].
<table>
<thead>
<tr>
<th>requirement</th>
<th>r_1</th>
<th>r_2</th>
<th>r_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>profit</td>
<td>effort</td>
<td>risk</td>
</tr>
<tr>
<td>s_1</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>s_2</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>s_3</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>average(AVG)</td>
<td>4.67</td>
<td>2.67</td>
<td>3.33</td>
</tr>
</tbody>
</table>

Table 1. Contribution (on a scale 1–10) of requirements \( R = \{r_1, r_2, r_3\} \) to the interest dimensions \( \text{Dim} = \{\text{profit, effort, risk}\} \).

Following the "aggregated utilities" approach, utility analysis determines a prioritization. AVG is one possible aggregation function – for further alternatives we refer to [4].

<table>
<thead>
<tr>
<th>importance(smp)</th>
<th>1</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>risk</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Importance of interest dimensions in a specific requirements prioritization context.

<table>
<thead>
<tr>
<th>r_1</th>
<th>r_2</th>
<th>r_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>utility(r_i)</td>
<td>32.66</td>
<td>58.34</td>
</tr>
<tr>
<td>prioritization</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. Utility and prioritization of individual requirements \( R = \{r_1, r_2, r_3\} \) with regard to the interest dimensions \( \text{Dim} = \{\text{profit, effort, risk}\} \). In this context, we assume that each requirement has its unique prioritization, i.e., prioritization(\( r_i \)) = prioritization(\( r_j \)) \( \rightarrow i = j \).

criteria such as maximum profit, maximum customer value, and minimum risk. For simplicity, Table 5 contains only binary constraints, however, these are generalizable to higher-order constraints [2], for example, coupling(\( \text{rel}_{ra}, \text{rel}_{rb}, \text{rel}_{rc} \)) would be translated into \( \text{rel}_{ra} \lor \text{rel}_{rb} \land \text{rel}_{rc} \).

An example of a constraint-based release configuration task is the following. This task includes the three requirements from Section 2. Furthermore, three releases are available.

- \( R = \{\text{rel}_{r1}, \text{rel}_{r2}, \text{rel}_{r3}\} \)
- \( \text{domain}(\text{rel}_{r1}) = [1..3] \)
- \( \text{PC} = \{ \text{pc}_1 : \text{rel}_{r2} \leq \text{rel}_{r1}, \text{pc}_2 : \text{rel}_{r2} \leq \text{rel}_{r3}, \text{pc}_3 : \text{rel}_{r1} \leq \text{rel}_{r3} \}\)
- \( \text{RC} = \{ \text{rc}_1 : \text{rel}_{r1} = \text{rel}_{r2}, \text{rc}_2 : \text{rel}_{r1} = 1 \}\)

Definition 3: Constraint-based Release Configuration: A constraint-based release configuration (solution) for a constraint-based release configuration task can be represented by a complete assignment \( \text{RP} = \{\text{rel}_{ra} = \text{ra}, \text{rel}_{rb} = \text{vb}, ..., \text{rel}_{rk} = \text{vk}\} \) where \( \text{vk} \) is the release number of requirement \( rk \) such that \( \text{RP} \cup \text{PC} \cup \text{RC} \) is consistent.

In the context of our working example, an example of a constraint-based release configuration is \( \text{RP} = \{\text{rel}_{r1} = 1, \text{rel}_{r2} = 1, \text{rel}_{r3} = 2\} \).

As it is often the case, prioritization as well as release planning is an iterative process [8]. As a consequence, prioritization \( PC \) of requirements change (resulting in \( PC' \)) as well as release constraints, i.e., \( RC \) is transformed into \( RC' \). In such contexts, situations can occur where \( \text{RP} \cup \text{PC}' \cup RC' \) becomes inconsistent and we are in the need of a reconfiguration of \( \text{RP} \) (resulting in \( \text{RP}' \)). Consequently, a release reconfiguration task has to be solved (see Definition 4).

Definition 4: Release Reconfiguration Task: A release reconfiguration task can be defined by a tuple \( (\text{RP}, \text{PC}', \text{RC}') \) where \( \text{PC}' \) represents a set of adapted prioritization constraints, \( \text{RC}' \) a set of adapted release constraints, and \( \text{RP} \cup \text{PC}' \cup RC' \) is assumed to be inconsistent. The underlying task is to identify a minimal set of constraints \( \Delta \subseteq \text{RP} \) such that \( \text{RP} - \Delta \cup \text{PC}' \cup RC' \) is consistent. If parts of \( \text{RP} \) have already been implemented, we can assume \( \text{RP} = \text{RP}_{\text{completed}} \cup \text{RP}_{\text{open}} \) and the task is to identify a diagnosis \( \Delta \) with \( \text{RP}_{\text{open}} - \Delta \cup \text{RP}_{\text{completed}} \cup \text{PC}' \cup RC' \) is consistent.

A reconfiguration for a given release reconfiguration task can be defined as follows (see Definition 5).

Definition 5: Release Reconfiguration: A release reconfiguration (solution) for a release reconfiguration task can be represented by an assignment \( \text{RECONF} = \{\text{rel}_{ra} = \text{ra}', \text{rel}_{rb} = \text{vb}', ..., \text{rel}_{rk} = \text{vk}'\} \) where \( \text{rel}_{vi} = \text{vi}' \in \text{RECONF} \rightarrow \text{rel}_{vi} = \text{vi} \in \Delta \) and \( \text{vi} \neq \text{vi}' \).

In this context, \( \Delta \) represents a diagnosis, i.e., a minimal set of constraints that, if deleted from \( \text{RP} \) (\( \text{RP}_{\text{open}} \)), help to restore consistency, i.e., \( \text{RP} - \Delta \cup \text{PC}' \cup RC' \) is consistent.

The set \( \Delta \) can be determined on the basis of a model-based diagnosis algorithm such as \( \text{FASTDIAG} \) [6] which returns a minimal set of constraints that have to be deleted in order to restore consistency. In order to assure the existence of a diagnosis \( \Delta \), we have to assume the consistency of \( \text{PC}' \cup RC' \).

One could also be interested in identifying minimal sets of changes \( \Delta \) in given prioritizations \( PC \) such that \( \text{PC} - \Delta \cup RC \) is consistent. Table 7 provides an overview of example (re-)configuration services that can be provided in release configuration scenarios. (1) proposed prioritizations (PC) have to be checked with regard to their consistency with a defined set of release constraints (RC). (2) Assuming the consistency of RC
Table 4. Contribution (on a scale 1–10) of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $Dim = \{profit, effort, risk\}$. High values for profit denote a high associated profit, whereas high values for effort and risk denote low associated effort and risk estimates.

Following the "aggregated predictions" approach, utility analysis determines stakeholder-specific prioritizations.

<table>
<thead>
<tr>
<th>requirement</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$s_2$</td>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$s_3$</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>constraint</th>
<th>formalization</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coupling($rel_{a}, rel_{b}$)</td>
<td>$rel_{a} = rel_{b}$</td>
<td>${rel_{a}, rel_{b}}$ have to be implemented in the same release</td>
</tr>
<tr>
<td>different($rel_{a}, rel_{b}$)</td>
<td>$rel_{a} \neq rel_{b} \lor rel_{a} = 0 \land rel_{b} = 0$</td>
<td>${rel_{a}, rel_{b}}$ have to be implemented in different releases</td>
</tr>
<tr>
<td>atleastone($rel_{a}, rel_{b}$)</td>
<td>$\neg{2}$</td>
<td>at least one out of ${rel_{a}, rel_{b}}$ has to be assigned to a release</td>
</tr>
<tr>
<td>atmostone($rel_{a}, rel_{b}$)</td>
<td>$\neg{2}$</td>
<td>at most one out of ${rel_{a}, rel_{b}}$ has to be assigned to a release</td>
</tr>
<tr>
<td>weakprecedence($rel_{a}, rel_{b}$)</td>
<td>$rel_{a} \leq rel_{b}$</td>
<td>$rel_{a}$ must be implemented before $rel_{b}$ or in the same release</td>
</tr>
<tr>
<td>strictprecedence($rel_{a}, rel_{b}$)</td>
<td>$rel_{a} &lt; rel_{b}$</td>
<td>$rel_{a}$ must be implemented before $rel_{b}$ or in the same release</td>
</tr>
<tr>
<td>valuedependency($rel_{a}, rel_{b}$)</td>
<td>$\neg{2}$</td>
<td>$rel_{a}$ and $rel_{b}$ must be implemented in nearly the same release</td>
</tr>
<tr>
<td>effortdependency($rel_{a}, rel_{b}$)</td>
<td>$\neg{2}$</td>
<td>$rel_{a}$ and $rel_{b}$ must be implemented in nearly the same release</td>
</tr>
<tr>
<td>fixed($rel_{a}$)</td>
<td>$rel_{a} = a$</td>
<td>requirement $r$ must be implemented in release $a$</td>
</tr>
<tr>
<td>notlaterthan($rel_{a}$)</td>
<td>$rel_{a} \leq a$</td>
<td>requirement $r$ must not be implemented after release $a$</td>
</tr>
<tr>
<td>notearlierthan($rel_{a}$)</td>
<td>$rel_{a} \geq a$</td>
<td>requirement $r$ must not be implemented before release $a$</td>
</tr>
</tbody>
</table>

Table 5. Example release constraints. For the constraint types valuedependency and effortdependency we assume a maximum deviation of $k$. 
and the inconsistency of $PC \cup RC$, a minimal set of elements in $PC$ has to be identified such that $PC - \Delta \cup RC$ is consistent. (3) Assuming the consistency of $PC' \cup RC'$, a minimal set of elements in $RP$ has to be identified (i.e., a potential change of the current release plan) such that $RP - \Delta \cup PC' \cup RC'$ is consistent. (4) Given a diagnosis $\Delta$ for $RP$ (with regard to $PC' \cup RC'$), a constraint solver can determine a solution for $RP - \Delta \cup PC' \cup RC'$. (5) If there exists a test case $t \in T$ with inconsistent ($t \cup PC$), a diagnosis $\Delta$ represents a minimal set of elements in $PC$ such that $PC - \Delta \cup t$ is consistent $\forall t \in T$.

Table 7. Overview of example release (re-)configuration services.

<table>
<thead>
<tr>
<th>ID</th>
<th>service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>consistency check of $PC \cup RC$</td>
</tr>
<tr>
<td>2</td>
<td>diagnosis of $PC$ with regard to $RC$</td>
</tr>
<tr>
<td>3</td>
<td>diagnosis of $PC$ with regard to test cases $T$</td>
</tr>
<tr>
<td>4</td>
<td>diagnosis of $RP$ with regard to $PC' \cup RC'$</td>
</tr>
<tr>
<td>5</td>
<td>reconfiguration of $RP$ with regard to $PC' \cup RC'$</td>
</tr>
<tr>
<td>6</td>
<td>diagnosis of $RC'$</td>
</tr>
</tbody>
</table>

### 4 Evaluation

Beside performance analyses, there are different alternative ways to evaluate the release planning related services mentioned in Table 7.

**Release plans** as outcome of a configuration process can be evaluated with regard to different interest dimensions such as profit, risk, and effort. The corresponding utility function is the following (see Formula 3).

$$utility(RP) = \frac{\sum_{rel_s \in RP} \sum_{d \in Dim} contrib(c,d) \times impl(d)}{|rel_s \in RP|}$$  

(3)

An evaluation of the utility of release plans generated with the Choco constraint solver\(^6\) is depicted in Table 8. A corresponding performance evaluation is depicted in Table 9. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints and a corresponding release plan 10 times, randomly changed 10% of the (original) release constraints and determined a reconfiguration (for the given release plan). We can observe a decreasing similarity with a corresponding increasing number of release constraints.

Table 8. Avg. utility of release plans without optimization.

| $|RC|$ | 1  | 5  | 10 | 25 | 50 | 100 |
|------|----|----|----|----|----|-----|
| 25   | 114.6 | 110.8 | 123.3 | 124.2 | 114.2 | 118.4 |
| 50   | 125.6 | 110.9 | 118.0 | 121.9 | 112.9 | 120.7 |
| 100  | 114.4 | 99.6 | 111.3 | 108.9 | 111.3 | 120.9 |
| 250  | 114.3 | 111.3 | 114.2 | 112.0 | 114.4 | 111.3 |
| 500  | 123.1 | 61.8 | 70.6 | 88.4 | 110.6 | 123.1 |
| 1000 | 112.3 | 63.1 | 50.7 | 71.7 | 95.3 | 109.4 |

Table 9. Avg. performance of release plan determination in ms.

Reconfigurations of release plans can be evaluated with regard to the similarity between the new configuration and the old configuration where $a(S)$ denotes the set of variable assignments contained in solution $S$.

$$similarity(S_{old}, S_{new}) = \frac{a(S_{old}) \cap a(S_{new})}{a(S_{old}) \cup a(S_{new})}$$  

(4)

An evaluation of the similarity between reconfigurations and original release plans is depicted in Table 10. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints and determined a reconfiguration (for the given release plan). We can observe a decreasing similarity with a corresponding increasing number of release constraints.

Table 10. Avg. similarity of reconfigurations.

| $|RC|$ | 1  | 5  | 10 | 25 | 50 | 100 |
|------|----|----|----|----|----|-----|
| 25   | 0.96 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| 50   | 0.93 | 0.97 | 0.99 | 1.00 | 1.00 | 1.00 |
| 100  | 0.87 | 0.96 | 0.97 | 0.99 | 0.99 | 1.00 |
| 250  | 0.96 | 0.87 | 0.93 | 0.98 | 0.99 | 0.99 |
| 500  | 0.99 | 0.76 | 0.86 | 0.95 | 0.97 | 0.99 |
| 1000 | 1.00 | 0.73 | 0.78 | 0.90 | 0.95 | 0.97 |

---

\(^6\) choco-solver.org
conservativism(Δ, C) = 1 − \frac{|\Delta|}{|C|} \quad (5)

relevance(Δ = \{δ_1, δ_2, ..., δ_q\}) = \frac{\sum_{\delta_i \in \Delta} rel(\delta_i)}{|\Delta|} \quad (6)

An evaluation of conservativism and relevance of generated diagnoses is depicted in Table 11. For each combination of |RC| \times \#releases, we randomly generated |RC| release constraints 10 times, assigned importance values to these constraints, and randomly changed 10% of the constraints. The resulting (inconsistent) constraint sets were diagnosed with FastDiag [6]. The corresponding evaluation results are depicted in Table 11. We can observe a decreasing degree of conservativism with an increasing number of release constraints RC.

| |RC| | 1 | 5 | 10 | 25 | 50 | 100 |
|---|---|---|---|---|---|---|---|
|25 |0.84/1.0|0.99/2|0.99/4|1.00/0|1.00/0|1.00/0|
|50 |0.75/1.0|0.99/5|0.99/6|1.00/0|1.00/0|1.00/0|
|100 |0.62/1.0|0.98/4|0.97/7|1.00/2|1.00/0|1.00/0|
|250 |0.58/1.0|0.86/1.0|0.96/1.0|0.99/9|1.00/8|1.00/1|
|500 |0.56/1.1|0.76/1.0|0.89/1.0|0.98/1.0|0.97/8|1.00/8|
|1000 |0.55/1.2|0.64/1.0|0.76/1.0|0.92/1.0|0.97/8|0.99/0|

Table 11. Avg. conservativism and relevance (c/r) of diagnoses.

5 Conclusion and Future Work

In this paper, we have shown how to represent release planning as a configuration problem. This representation is a major basis for supporting re-planning tasks, i.e., the adaptation of plans that become inconsistent due to changing constraints (e.g., a changing availability of resources). In this context, we also focused on introducing concepts that support the automated adaptation of release plans (reconfiguration of release plans) and different variants thereof such as the diagnosis of release constraints and prioritization constraints. To show the applicability of the presented concepts, we have presented initial evaluation results based on a couple of evaluation metrics.

Future work will include the development and evaluation of different types of preference aggregation functions (see Section 2) with regard to their capability of generating relevant prioritizations. Furthermore, we will optimize the determination of release plans, reconfigurations, and diagnoses by integrating intelligent search heuristics that help to improve the quality of identified solutions. In this context, we will compare constraint-based reasoning approaches with local search based ones [8] with regard to efficiency and solution quality.

Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OpenReq (732463).

REFERENCES


