A Hybrid Release Planning Method and its Empirical Justification

Mark Przepiora
University of Calgary
Calgary, Alberta, Canada
mark.przepiora@ucalgary.ca

Reza Karimpour
University of Calgary
Calgary, Alberta, Canada
reza.karimpour@ucalgary.ca

Guenther Ruhe
University of Calgary
Calgary, Alberta, Canada
ruhe@ucalgary.ca

ABSTRACT

Background: The use of Constraint Programming (CP) has been proposed by Regnell and Kuchcinski to model and solve the Release Planning Problem. However, they did not empirically demonstrate the advantages and disadvantages of CP over existing release planning methods.

Aims: The aims of this paper are (1) to perform a comparative analysis between CP and ReleasePlanner (RP), an existing release planning tool, and (2) to suggest a hybrid approach combining the strengths of each individual method.

Method: (1) An empirical evaluation was performed, evaluating the efficiency and effectiveness of the individual methods to justify their hybrid usage. (2) A proof of concept for a hybrid release planning method is introduced, and a real-world dataset including more than 600 features was solved using the hybrid method to provide evidence of its effectiveness.

Results: (1) Use of RP was found to be more efficient and effective than CP. However, CP is preferred when advanced planning objectives and constraints exist. (2) The hybrid method (RP&CP) greatly outperformed the individual approach (CP), increasing computational solution quality by 87%.

Conclusion: We were able to increase the expressiveness and thus applicability of an existing, efficient and effective release planning method. We presented evidence for its computational effectiveness, but more work is needed to make this result significant.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*tools*; D.2.9 [**Software Engineering**]: Management.

General Terms

Algorithms, Performance

Keywords:

Release planning, efficiency of use, user satisfaction, hybrid algorithm, performance evaluation

ESEM'12, September 19–20, 2012, Lund, Sweden.

1. INTRODUCTION

Release planning is the problem of assigning features to subsequent releases in consideration of technological and resource constraints. The objective of planning is to find the most attractive composition of features delivered in releases to users and customers.

A variety of methods and techniques do exist to approach the various formulations of the release planning problem. A systematic literature review was conducted by Svahnberg at al. [1]. More recently, Regnell and Kuchcinski [2] proposed a solution for modeling the software release planning problem as a constraint satisfaction problem. The related solution method called Constraint Programming (CP) is a programming approach in which the state of discrete variables and their relationships are controlled using constraints [3]. Compared to procedural programming, constraints do not specify any sequence of actions, but rather the properties of a solution to be found. This is similar to declarative programming. Their CP formulation of the release planning formulation originally introduced in [4] includes relative and absolute stakeholder priorities among features, interdependencies, and release-specific resource constraints.

The results of Regnell and Kuchcinski [2] served as a proof-of concept to show the applicability of CP for a broad range of release planning formulations. Two main follow-up questions were derived from there: (i) How applicable is the formal notation used in CP from an application and user's perspective? (ii) How is the scalability of the approach to solve problems with hundreds of features?

To approach these questions, a comparative analysis was conducted between two solution approaches:

- CP: Constraint programming as implemented in MiniZinc
- RP: Formulation of release planning as integer linear programming problem and usage of special-purpose solvers as described in [5]

In this short paper, two research questions (RQ) are tackled:

RQ1: Perform an empirical analysis to compare CP against RP in terms of efficiency of use and user satisfaction.

RQ2: From the results of RQ1, combine the respective strengths of CP and RP and offer them in an integrated method.

In Section 2, we briefly report the results from two empirical studies comparing CP versus RP. As a result of the two studies, we propose a hybrid solution method for release planning called RP&CP. The hybrid method described in Section 3 is a two-staged solution approach which combines the higher flexibility in problem formulation (in terms of describing objectives and constraints) of CP with the advantages offered by RP. The results of the proof-of-concept analysis of RP&CP are given in Section 4. Outlook for future research is the content of Section 5.

2. COMPARATIVE ANALYSIS OF TWO RELEASE PLANNING METHODS

2.1 CONTEXT

The goal of our comparative analysis of two release planning methods with two associated tool implementations was to understand the principal advantages and disadvantages of each approach.

The tool used for CP is MiniZinc [2] which is a subset of a constraint modeling language called Zinc. In terms of complexity, MiniZinc is adequately high-level to formulate the constraint problems with reasonable effort. On the other hand it is simple enough to be adopted by existing solvers consistently.

The tool used for RP is ReleasePlanner [6], a web-based decision support system for release planning. The underlying solution method is based on algorithms (branch and bound, linear programming) combined with heuristics, all customized to the special structure of the problem formulation [5]. The RP problem formulation is limited to coupling and precedence constraints between features, linear resource constraints, and a (weighted) additive utility function aggregating stakeholder scores to features, given for a flexible number of planning criteria. While this formulation has been sufficient for solving a number of real-world cases, it appears desirable to enhance modeling capabilities both for the planning objectives (e.g., non-linear objectives) and constraints (e.g., general logical constrains such as "at least one feature from a given candidate set").

2.2 TWO EMPIRICAL STUDIES

Two empirical studies were conducted to compare two different aspects of the usages of CP versus RP. We do not intend to make a preference decision between the two systems. Instead, we were aiming to find out which approach (method and associated tool) is preferable in which situation. The preference criteria focused on are efficiency (referring to usage of resources) and effectiveness (referring to quality of solutions in dependence of effort spent).

2.2.1 EFFICIENCY

In [7], efficiency is considered to be a component of the usability of a system. It is defined as the capability of the software system to empower users to allot reasonable amounts of resources in relation to the effectiveness gained in a specified context of use. To measure the efficiency, eight frequently occurring tasks of the release planning process were considered:

- Adding a feature to feature repository (Task 1)
- Editing a feature already available (2)
- Defining a dependency between features (3)
- Defining available resources (4)
- Editing available resources (5)
- Inputting stakeholder votes (6)
- Editing stakeholder importance (7)
- Generating a solution (8)

In addition, we looked at

- Number of errors or failed commands and (9)
- Frequency of help or documentation use (10)

Experimental subjects were asked to perform the planning process for a benchmark problem introduced in [4]. Seven participants were recruited. The results of the measurements are summarized in Table 1. The unit of measurement for tasks 1 to 8 is *seconds* (average) while the numbers for task types 9 and 10 represent *frequencies* (average).

Table 1. Efficiency evaluation results

	1	2	3	4	5	6	7	8	9	10
CP	186	154	135	147	197	154	161	147	3	6
RP	134	61	102	73	61	98	65	95	1	1

2.2.2 EFFECTIVENESS

In evaluating the effectiveness of each approach, we are interested only in exploring the computation that generates a release plan for a completely-defined project. The metric we use is best solution quality (that is, utility function value divided by maximum value) after 15 seconds of runtime, which we denote $q \in [0, 1]$.

Ideally, we would like to evaluate the performance of each approach as a function of the size of the input. However, the size of an RP project is determined by a number of parameters, some of which may have a greater impact on performance than others.

Our goal is to explore this space of parameters as fully as is feasible, and in particular we identified 7 parameters of importance: the number of planning items (N); the number of resources (M); the number of release periods (K); the number of weak precedence constraints (S); the number of strict precedence constraints (H); the number of pre-assignment constraints (L); and the "resource tightness" (T) of the project, that is, the ratio of sum total resource consumptions to sum total resource capacities.

We created a data set of 96 projects by creating a project of every possible combination of parameter values from the following list:

N = {30, 150}, M = {1, 5}, K = {1, 5}, S = {N/25, N/10}, H = {N/10}, L = {N/50, N/10}, T = {0.2, 1.0, 3.0}.

Item values were chosen uniformly from the interval [1000, 9000] and releases were weighted in decreasing order. Resource consumptions and resource capacities were chosen from uniform distributions as to result in the target resource tightness. Precedence constraints were generated from iteratively choosing random constraints. At each iteration, a cycle detection algorithm was applied to ensure that no logical contradictions were generated during problem generation.

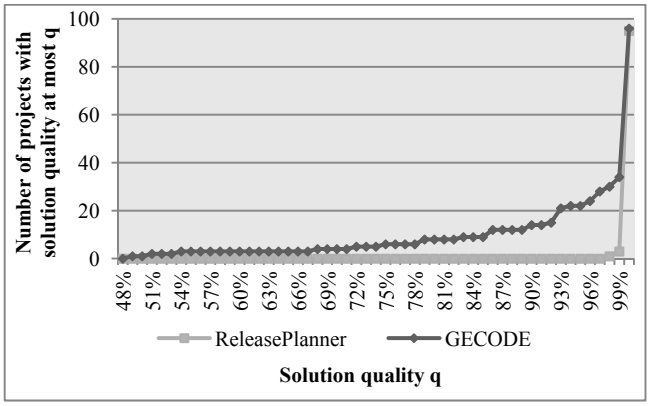

Figure 1. Cumulative frequency graph of solution quality.

In order for CP to be used to optimize the project as a whole, the RP parts must be converted into MiniZinc as well. To do so, we developed and implemented an RP to MiniZinc transformation in the form of a Java application that converts an input RP project into an equivalent CP in the MiniZinc format, following logical definitions that we briefly describe in Section 3.1.

The resulting CP may be solved using any MiniZinc solver, but in particular we considered GECODE[1], an open-source constraint solver which, according to the creators of MiniZinc, is currently the best-performing MiniZinc solver available.

We pre-computed the optimal solution for each project during previous research, and kept this information to allow us to determine the solution quality achieved by each solver. We solved each project using both solvers with 15 second time-limits using a 2.8GHz CPU and 4GB RAM, and recorded the utility values of the best solutions found by each solver within the allotted time.

Figure 1 shows a cumulative quality frequency graph for the solutions found by each solver. It can be seen that RP significantly outperformed GECODE in these tests, being able to solve all but 3 projects in the dataset to an optimality of at least 99% within the allotted time. GECODE was unable to do so in 34 cases, and in 14 cases achieved an optimality of less than 90%.

2.3 LIMITATIONS AND CONCLUSIONS

The results of the two empirical studies are not intended to provide a comprehensive and final comparison, but to motivate follow-up research. Users found RP easy to use while CP had mixed evaluations (for details, one may refer to [8]). Scores for RP showed that users can complete the intended task faster without extra help or training while getting started with CP needs some prior training. RP appeared to be clearly more effective, i.e., generating solutions of significantly higher degree of optimality.

Even in consideration of the existing threats to validity (including the small number of subjects having participated), there exists a clear pattern of usage of the two alternative systems. RP appears to be more efficient and effective in its use. It achieves high user satisfaction (which is confirmed from industrial users outside the survey made).

However, CP is the recommended preference in case of more sophisticated planning problems with more advanced planning objectives and constraints. Evaluation of real world software has shown that in addition to intrinsic constraints provided by classic feature models, many projects use custom constraints to model the intended domain [9]. CP has extensive and flexible support for defining relations and constraints between features. This makes it promising for industrial use while RP only supports two primary constraints, precedence and coupling that would not be sufficient for handling an industry level project like Linux.

3. A HYBRID RELEASE PLANNING METHOD

3.1 PROBLEM FORMULATION

The Release Planning Problem RPP can be described in an abstract way as the problem of defining an assignment (release plan) x = (x(1) … x(N)) of a given feature set F ={f(1) …f(N)} with the following meaning.

- x(n) = k if and only if feature f(n) is assigned to release k
- x(n) = 0 if feature f(n) is not offered at all

This assignment x is supposed to satisfy the following conditions.

- A utility function G(x) is maximized.
- The assignment x satisfies a set X of constrains.

RP allows a weighted linear function for G(x), which is composed from stakeholder scores related to different criteria. The constraint set X^{RP} in RP allows couplings between features, pre-assignment of features, linear resource constraints as well as soft and strict precedence constrains (for details we refer to [5]). We call such constraints "RP constraints".

CP is not limited and allows any logical constraint as well as any formulation of objectives. We call such constraints "Non-RP constraints". Below we list a few examples of non-RP constraints that are easily-expressible using CP.

- Mutual exclusion, i.e. constraints of the form, "exactly one of feature a or feature b must be offered".
- Additive synergy between features, e.g. "if all features in the set A are offered, then the value of the plan is increased by C points".
- Productivity investments, e.g. "if feature a is completed in release k, then all resource consumption is lowered by C% in release $k+1$ onward."

Such constraints motivate the following, hybrid approach, in which RP is extended with a module that allows project managers to implement non-RP constraints in their projects using a reusable template system.

3.2 CONSTRAINT DEFINITION

We have developed an expressive, domain-specific language based on MiniZinc used to define *templates* for commonly-used classes of constraints. Project managers may then *instantiate* these templates using a GUI without the need to comprehend the underlying code.

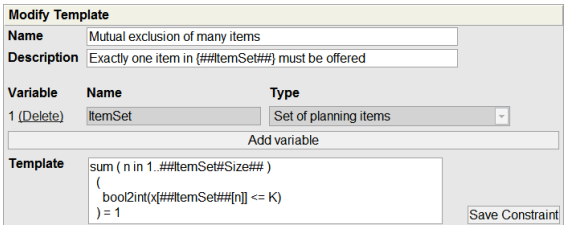

Figure 2. Template definition UI

This method enables the full expressive power of MiniZinc within RP without sacrificing usability for the project manager, since coding need only be performed once for each class of constraint that any manager wishes to use in the future.

We refer to our technical report [8] for a detailed usage scenario.

3.3 OPTIMIZATION

If the project manager does not choose to implement any non-RP constraints in his project, then the project resulting from this process is simply an RP project, and optimization can be performed using the existing, highly-effective RP solver.

But if the manager did add non-RP constraints to the project, then a hybrid algorithm is used. First, the relaxed problem is solved (ignoring the constraints outside X^{RP}) using the existing RP solver. Although the solution computed may in general violate the non-RP constraints of the project, we will use it as part of the CP search strategy as a starting solution.

To do so, the entire project, an RP project together with non-RP constraints, must be transformed into an equivalent CP expressed in the MiniZinc language. Non-RP constraints defined using the process in Section 3.2 are transformed naturally, as the template language is designed to easily translate into MiniZinc. On the other hand, in order to transform the remaining, RP-only portion of the project to MiniZinc, we use the tool we developed in Section 2.2.2.

[1] http://www.gecode.org/

Finally, parameters which instruct the MiniZinc solver to use the assignments from the RP solution as a starting point in its search strategy are added to the code. The project is then solved using a CP solver (as in Section 2.2.2, we used the GECODE solver).

4. PROOF-OF-CONCEPT ANALYSIS

To demonstrate the applicability of our hybrid method, we have used a real-world problem studied in [10], an RPP which includes 633 features.

Step 1: We began by computing release plans for the project using the RP solver. We saved the best solution found after 15 seconds. This solution had a 99.94% degree of optimality. We refer to this solution as the relaxed solution.

Step 2: We converted the RP project to the MiniZinc format using our tool, and added the following five non-RP constraints and objectives to the project in addition to its existing constraints:

- *At most one* of items 1, 2 and 3 may be released.
- *At least two* of items 9, 10 and 11 must be released.
- *Exactly two* of items 12, 13 and 14 must be released.
- If all of items 6, 7 and 8 are offered, then the utility function value is increased by 12,000 points (*non-linear utility G(x)*).
- If all of items 15, 16 and 17 are offered, then the utility function value is increased by 12,000 points (*non-linear utility function G(x)*).

We verified that the relaxed solution computed in Step 1 violated the first and third constraints above, which means that the CP solver may not simply reuse the relaxed solution as a solution to the non-RP constraints.

Step 3: Next, we computed solutions for this resulting project in two ways: (1) First, we used the CP solver alone, without modifying its default search strategy. (2) Second, we modified the search strategy to coerce the solver into using the relaxed, RP solution as a starting solution for optimization.

Table 2. Solution quality, CP-only vs. RP&CP

Method	Solution quality			
	5 seconds	15 seconds	5 minutes	30 minutes
CP	52.13%	52.89%	52.94%	52.94%
RP&CP	91.16%	98.83%	98.83%	98.83%

We computed solutions using 5-second, 15-second, 5-minute and 30-minute time limits on a personal computer with the same specifications as in Section 2.2.2, and recorded the utility function value of the best solution found during each run.

Step 4: The results of these computations are summarized in Table 2. We see that using CP alone achieved a solution quality of 52.89% after 15 seconds (improved only to 52.94% after 30 minutes), while the hybrid RP+CP computed a near-optimal solution after 15 seconds. This corresponds to an improvement of about 87%.

5. SUMMARY AND CONCLUSIONS

The underlying conjecture of this short paper was that a combination of two existing release planning would result in an even stronger (hybrid) approach. We performed two independent empirical investigations to evaluate the relative strengths and weaknesses of the two methods under investigation. We reported the design and implementation of the hybrid approach. From applying user-friendly templates, advanced logical constraints and non-linear planning functions could be easily expressed.

The results of the proof of concept evaluation are taken as first evidence that our conjecture is true. We were able to increase the expressiveness and thus applicability of an existing, efficient and effective release planning method RP. For future work, we plan to perform a broad evaluation of the hybrid approach with problems of varying size and structural characteristics. While RP has been industrially evaluated, we plan to do the same for the hybrid approach. Besides academic interest, the motivation to look into higher expressiveness came also from industrial applications. We will also look into the applicability of genetic search algorithms directly applied on top of RP. This would further simplify the process and eliminate the dependency from an additional solver.

6. ACKNOWLEDGMENT

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12. And thanks to all participants of the empirical studies.

7. REFERENCES

[1] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. B. Saleem, and M. U. Shafique, "A systematic review on strategic release planning models," *Information and Software Technology*, vol. 52, no. 3, pp. 237-248, 2010.

[2] B. Regnell and K. Kuchcinski, "Exploring Software Product Management decision problems with constraint solving-opportunities for prioritization and release planning," in *Fifth International Workshop on Software Product Management (IWSPM)*, 2011, pp. 47-56.

[3] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier Science, 2006.

[4] G. Ruhe and M. O. Saliu, "The Art and Science of Software Release Planning," *IEEE Software*, vol. 22, no. 6, pp. 47-53, Nov. 2005.

[5] A. Ngo-The and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning," *Soft Computing*, vol. 12, no. 1, pp. 95-108, Aug. 2007.

[6] "ReleasePlanner." [Online]. Available: www.releaseplanner.com. [Accessed: 20-May-2012].

[7] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda, "Usability measurement and metrics: A consolidated model," *Software Quality Journal*, vol. 14, no. 2, pp. 159-178, 2006.

[8] M. Przepiora, R. Karimpour, and G. Ruhe, Constraint Programming versus Specialized Tool Support: A Comparative Analysis for the Release Planning Problem," May 2012. [Online] Available: http://pages.cpsc.ucalgary.ca/~przepiom/esem2012/ [Accessed: May 20, 2012].

[9] T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki, "Variability modeling in the real: a perspective from the operating systems domain," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 73-82.

[10] S. bin Saleem, Y. Yu, and B. Nuseibeh, "An Empirical Study of Security Requirements in Planning Bug Fixes for an Open Source Software Project." TR 2012/01, Dep. of Computing, Faculty of Mathematics, Computing and Technology, The Open University, London, UK, 2012.