A Modified Indicator-Based Evolutionary Algorithm (Mibea) AND SUMO Algorithm In Remodularidation

G Srinivas Achar[1], Vijay Swaroop[2], Vinutha Yadav[3], Swati Kumari[4]
[1, 2]Assistant Professor
[1, 2, 3, 4,] Atria Institute of Technology, Visvesvaraya Technological University, Bangalore

Abstract- *Remodularising the components of a software system is challenging: sound design principles (e.g., coupling and cohesion) need to be balanced against developer intuition of which entities conceptually belong together. Despite this, automated approaches to remodularisation tend to ignore domain knowledge, leading to results that can be nonsensical to developers. Nevertheless, suppling such knowledge is a potentially burdensome task to perform manually. A lot information may need to be specified, particularly for large systems. Addressing these concerns, we propose the SUpervised reMOdularisation (SUMO) approach.Multi-objective evolutionary algorithms (MOEAs) based on the concept of Pareto-dominance have been successfully applied to many real-world optimisation problems. Recently, research interest has shifted towards indicator-based methods to guide the search process towards a good set of trade-off solutions. One commonly used approach of this nature is the indicator-based evolutionary algorithm (IBEA). In this study, we highlight the solution distribution issues within IBEA and propose a modification of the original approach by embedding an additional Pareto-dominance based component for selection. The improved performance of the proposed modified IBEA (mIBEA) is empirically demonstrated on the well-known DTLZ set of benchmark functions. Our results show that mIBEA achieves comparable or better hypervolume indicator values and epsilon approximation values in the vast majority of our cases (13 out of 14 under the same default settings) on DTLZ1-7. The modification also results in an over 8-fold speed-up for larger populations. SUMO is a technique that aims to leverage a small subset of domain knowledge about a system to produce a remodularisation that will be acceptable to a developer. With SUMO, developers refine a modularisation by iteratively supplying corrections. These corrections constrain the type of remodularisation eventually required, enabling SUMO to dramatically reduce the solution space. This in turn reduces the amount of feedback the developer needs to supply. We perform a comprehensive systematic evaluation using 100 real world subject systems. Our results show that SUMO and mLBEA guarantees convergence on a target remodularisation with a tractable amount of user interaction.*

Keywords- Software remodularisation, domain knowledge, set partitioning.

I. INTRODUCTION

REMODULARISATION remains a difficult and unsolved problem in software maintenance. As software evolves to meet new requirements, its design invariably deteriorates,makingithardertomaintain.Tomoderatethisdeterioration, systems can be remodularised so that their components are configured in a way that enables, for example, comprehension [1] or performance [2]. The task of restructuring a system by hand tends to be prohibitively time consuming andresourceintensive[3]. Existing remodularisation algorithms have sought to produce improved designs automatically, but have been unable to do so satisfactorily [3], [4]. Automated algorithms have focused on using techniques such as clustering or formal concept analysis [5], [6], [7], [8], [9], [10], [11] to produce groupings for software components. However, these techniques formulate modules by focussing on the source code alone, and consequently tend to produce solutions that do not make sense from a conceptual point of view [3], [4], [12]. For example, an industrial study [3] of the popular Bunch remodularisation tool [11] found that Bunch's results were "non-acceptable for the domain experts" for the task of reorganising software components for a large medical system, consisting of several million lines of code. In multi-objective optimisation, where multiple objectives are optimised simultaneously, the goal is to find a set of Pareto-optimal solutions known as the Pareto front (PF). The PF consists of a set of solutions that are not dominated by each other, which are termed as non-dominated solutions, representing the trade-off that exists between different objectives. This dominance relation, also known as Pareto dominance relation, ($\prec$) is defined between solutions x1 and x2. We say w.l.o.g., in a minimisation problem that x1 dominates x2 ($x1 \prec x2$) if and only if $fi(x1) \leq fi(x2)$ for all k objective functions ($i \in \{1,...k\}$), and $fi(x1) < fi(x2)$ for at least one objective function. One of the difficulties in multi-objective search is to find a set of solutions to minimise the distance to the true Pareto front (PF) while maintaining the diversity of the solution set in the objective space. Multi-objective evolutionary algorithms (MOEAs) are widely used to solve various multi-objective optimisation problems [16] and are considered to be general and robust search mechanisms

[1]. Evolutionary algorithms (EAs) are a class of stochastic optimisation methods that mimic the process of evolution in nature. Examples of EAs, such as genetic algorithms, evolutionary programming, and evolution strategies [1] operate on a set of solutions using the basic principles of natural evolution: selection, reproduction by means of recombination and mutation. The algorithmic difference between single-objective EAs and MOEAs is that additionally in an MOEA, the multiple objectives of a solution must be transformed into a single fitness value to facilitate the comparison of individual solutions [6]. Thus, MOEAs often vary in the method of this transformation, considering the balance of convergence and diversification during the search. Some MOEAs, such as NSGA-II [7], SPEA2 [18], and AGE [3, 13] incorporate Pareto-based ranking of the individuals and an additional density measurement (crowding distance in NSGA-II, k-th nearest neighbour in SPEA2, dominance in AGE) in the objective space. However, between two non-dominated solutions, purely Pareto-based MOEAs are not able to ascertain which solutions have better potential for convergence. IBEA[17] was one of the earliest indicator-based MOEAs proposed in the literature. Originally, it came in two variants: one using -dominance for guidance, denoted IBEA, and another based on hyper volume, denoted IBEAHD ('HD' stands for hyper volume difference) which will be referred to as IBEA from this point onward in this study. IBEA associates a fitness value with each solution based on the selected indicator (hyper volume or), attempting to guide the search towards the true PF. IBEA was shown to achieve significantly better performance on various benchmark functions than NSGA-II and SPEA2 [17], however the distribution of the solutions found by IBEA has rarely been reported or discussed in detail. In this paper, we propose a modified variant of IBEAHD, termed mIBEA, which adds a Pareto-based element to this indicator-based method, analysing the distribution of nondominated solutions found. For further information about MOEAs and indicator-based MOEAs in particular, we refer the interested reader to [5, 6, 16]. The remainder of the paper is structured as follows. We first describe the original IBEA in Section II-A, then present observations of the solution distributions observed using existing MOEAs in Section II-B. The proposed mIBEA is introduced in Section III. Experimental results comparing IBEA and mIBEA are presented in Section IV. These limitations have led to the realisation that software remodularisation techniques must necessarily involve a degree of input from an expert. Accordingly, several variants of existing modularisation algorithms have been developed, which seek to accommodate this need [11], [13]. However, they tend to be limited in practical terms as they either (a) interrogate the user for feedback in a way that renders them prohibitively expensive, or (b) fail to provide guidance to the user, leaving them with no indication of how much input is necessary or of value to the underlying algorithm. In order to address these problems, we introduce the Supervised reMOdularisation (SUMO) technique [14]. SUMO is based on the observation that existing general purpose clustering algorithms can be improved with relatively little domain knowledge [15]. Remodularisation algorithms often produce partialsolutions[3],[4],but given a set of corrections, these partial solutions may be transformed into desired modularisations. For example, given a proposed clustering for a data processing framework, a developer might make an observation that contradicts the current proposed solution, such as "Classes XMLParser and Abstract Parser belong together, but neither should be in the same module as Data Visualizer." SUMO provides a mechanism by which to enable the developer to feed-in this corrective information in the form of specific relationships, for example "XML Parser does not belong with Data Visualizer".

II. INDICATOR-BASED EVOLUTIONARY ALGORITHM AND SUMO ALGORITHM

Since the focus in the paper is on the hypervolume variant of IBEA, i.e. IBEAHD, we first give a detailed description of the original IBEA. We then provide visualisations and observations for the non-dominated solution sets found using IBEA and two other existing MOEAs, one Pareto dominance-based (NSGA-II [7]) and one indicator-based (SMS-EMOA [2]). A. Description of IBEA The core idea of IBEAHD is to employ a binary hypervolume indicator in the selection process, when determining which solutions survive to the next generation. The binary hypervolume indicator assigns a real-valued number to two solution sets with respect to a reference point. The formula of IHD(A,B) is defined as space that is dominated by population B, but not by A, shown in Equation (1) [17].

$$IHD(A,B) =(IH(B)-IH(A),\forall x2 \in B \exists x1 \in A : x1 \prec x2\ IH(A + B)-IH(A),o.w.\ (1)$$

where IH(A) denotes the hypervolume formed by the solution set A. Correspondingly, IH(A + B) means the hypervolume of the union of solution set A and B. IHD(A,B) is negative if all solutions in B are dominated by solutions in A. Note that IH(A) 6= IH(B).

The pseudocode of the original IBEA is given in Alg. 1. IBEA first randomly generates an initial population in Step1, then the following steps loop until the stopping criterion is satisfied. The objective values are scaled and the fitness is assigned to each individual in Step2 and Step3. Step4 performs environmental selection, iteratively removing the worst individual in the population P based on indicator value until µ individuals remain (this step will do nothing in the first iteration of the algorithm as P = µ). Upon removal of each solution, the indicator values of the remaining solutions must be updated. This step continues until the number of solutions in P does not exceed µ. The standard mating selection (Step6) and variation (Step7) steps are performed to generate new individuals and add them to the population P.

Fig. 2 summarises the steps of the SUMO algorithm, which we elaborate in more detail in Algorithm 1. The algorithm relies on two types of information that influences the solutions it generates:

1) An initial modularisation, Mod
2) Corrections Relþ and Rel

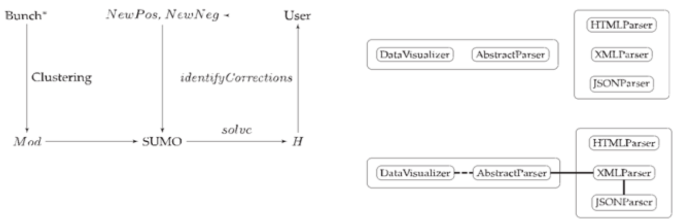


Fig. 2. The SUMO process: Starting with an initial modularisation Mod,SUMO repeatedly presents the hypothesised solution H and refines it according to relations supplied by the user. In this example Bunch(highlighted by the asterisk) generates the starting point; in practice other remodularisation algorithms can be used.

The algorithm begins with an initial modularisation Mod, which may be the current package structure, or a proposed modularisation from a tool such as Bunch [11]. The algorithm builds on this solution by iteratively soliciting feedback from a user and applying a constraint solver to produce new solutions.

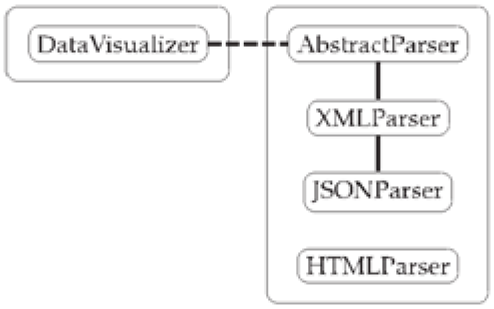


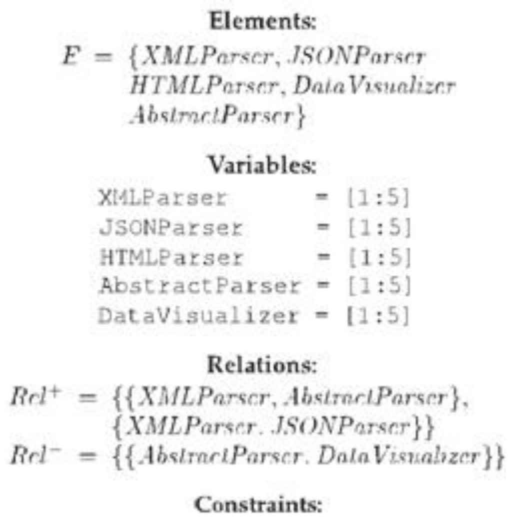
Fig. 3. Eliciting relations

```
Algorithm 1. SUMO Algorithm

Input: Mod
Data: Rel ; Rel$^{þ}$; solved; H; NewPos; NewNeg

Uses: solveðX; Y; ZÞ, identifyCorrectionsðXÞ
Result: H
Rel$^{þ}$      ;;
Rel       ;;

solved    false;

H         Mod;

while ð:solvedÞ do

ðNewPos; NewNegÞ identifyCorrectionsðHÞ; if ðNewPos [
NewNeg ¼ ;Þ then
    solved    true;
    else
    Rel$^{þ}$      Rel$^{þ}$ [ NewPos;
    Rel       Rel  [ NewNeg;
H         solveðRel$^{þ}$; Rel ; HÞ;
end

end

return H
```

III. METHEDOLOGY

Producing a partition of elements that is consistent with the relations in Rel^{þ} and Rel is a constraint satisfaction prob-lem that existing solvers can provide solutions for. Given a set of elements E ¼ fe$_0$; . . . ; e$_n$ g in a system, we represent each distinct module to which they potentially belong as a unique number drawn from the set N ¼ ½1 : n&. The solver must then find a set of assignments (i.e., a partition) p : E ! N, where each element in E is mapped to a number that denotes its module. The constraints on the possible mappings of p are contained within Rel^{þ} and Rel .

A pair fe$_i$; e$_j$g in Rel^{þ} implies pðe$_i$Þ ¼ pðe$_j$Þ. Similarly, the presence of a pair fe$_i$; e$_j$g in Rel implies that pðe$_i$Þ ¼6 pðe$_j$Þ. Fig. 4 shows an example in which the elements of the set E are to be remodularised subject to the given relations in Rel^{þ} and Rel . The lower portion of the figure shows how these can be translated into a constraint program that can be solved by existing constraint solvers.



Elements:

$$E = \{XMLParser, JSONParser\ HTMLParser, DataVisualizer\ AbstractParser\}$$

Variables:

```
XMLParser      = [1:5]
JSONParser     = [1:5]
HTMLParser     = [1:5]
AbstractParser = [1:5]
DataVisualizer = [1:5]
```

Relations:

$$Rel^{+} = \{\{XMLParser, AbstractParser\}, \{XMLParser, JSONParser\}\}$$
$$Rel^{-} = \{\{AbstractParser, DataVisualizer\}\}$$

Constraints:

Fig 4. . Representing remodularisation as a constraint program. Ele-ments (left) become variables, and user-supplied relations (right) are translated into constraints.

Firstly, we can see that the solutions from NSGA-II can cover the whole fronts of all the problems DTLZ1-3. Secondly, SMS-EMOA can reach almost full coverage of the PF for DTLZ1, however, on DTLZ2 and 3 the PFs have clear gaps between the border and central area. Thirdly, similar to SMSEMOA, IBEA also shows clear gaps (empty space) on the PF of DTLZ2. Also, solutions by the original IBEA are heavily concentrated in the corner points on DTLZ1 and DTLZ3. The GS measurement of the solution fronts obtained from NSGA-II, SMS-EMOA and IBEA (as illustrated in Figure 1) is shown in Table

I. The best GS on each benchmark function is highlighted in bold. Not surprisingly, IBEA performs poorly w.r.t. GS among all the three algorithms on DTLZ13. However, interestingly, although SMS-MOEA shows gaps on the resultant PFs of DTLZ2 and 3, the GS indicator still shows that it performs best on both problems. This suggests that under certain situations (e.g., PFs of DTLZ2 and 3 from SMS-EMOA) GS cannot capture the distribution of the PF, i.e., the spread calculated is not equivalent to the distribution.

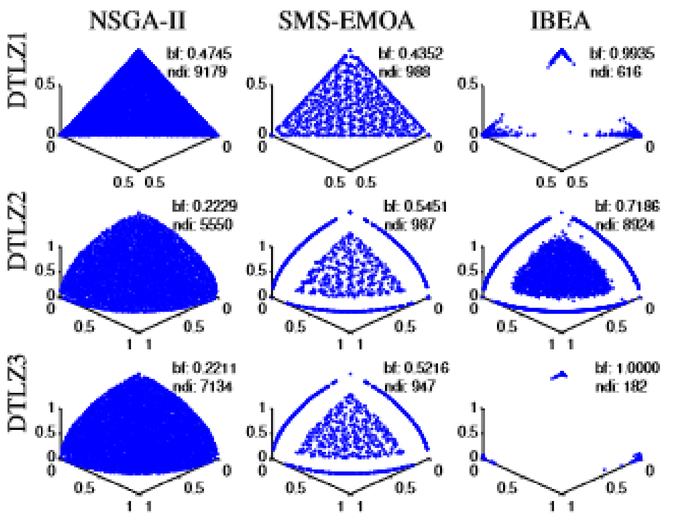
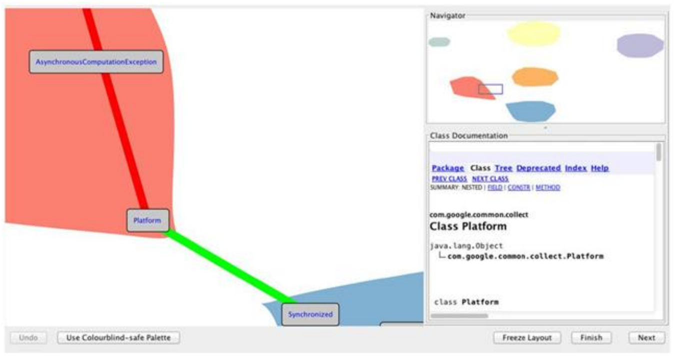


Fig 5: : Comparison of non-dominated solution sets for NSGAII, SMS-EMOA and IBEA.



Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD
com.google.common.collect
Class Platform
java.lang.Object
└ com.google.common.collect.Platform
class Platform

Fig. 6. The SUMO user interface, showing an overview of a hypothesised modularisation in the top right hand corner. The main panel zooms into a smaller part of this modularisation, allowing the user to add positive and negative information. SUMO aids the user by with the help of JavaDocs in the bottom right panel. In this example, the main panel shows a positive link between "AsynchronousComputationException" and "Platform", and a negative link between "Synchronized" and "Platform".

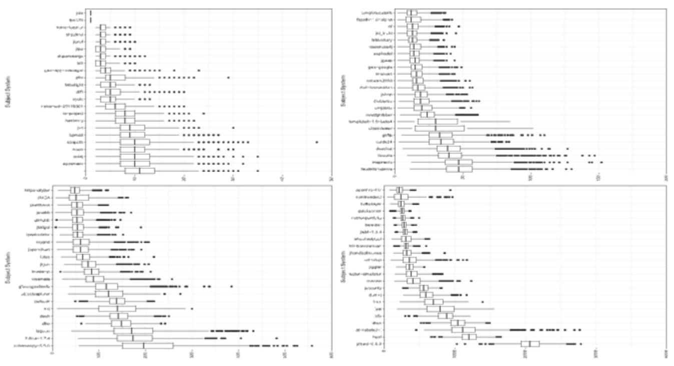


Fig. 7. Box plot of the number of SUMO iterations for convergence for each subject system for the 75 percent user

model, showing the variance for each subject system. The breaks are to illustrate scale.

Algorithm 2:

```
mIBEA Input: μ: population size;
N: number of total solution evaluations;
ρ: objective values scaling factor;
κ: indicator value (hypervolume difference) scaling factor
Output: A: Pareto set approximation
Step1: Initialisation (see Step 1 in Alg. 1);
Step2.1: Use the fast non-dominated sorting of NSGA-II to get non-dominated solutions in P and use the non-dominated solutions as the new P.
1) rank the solutions in P: Ranking rankedP = new Ranking(P);
2) get the non-dominated solutions:
P = rankedP.getSubfront(0);
Step2.2: Scale objective values (see Step 2 in Alg. 1); Steps 3-7 are the same as the original IBEA in Alg.
```

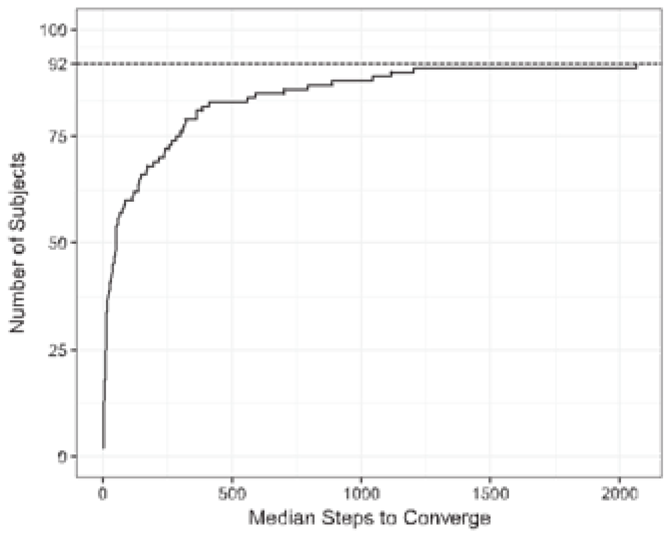


Fig. 8. Cumulative number of subject systems ordered by median steps to converge. Half of the SF100 is remodularised within 51 SUMO itera-tions; most are remodularised within 500. The dashed line indicates the total number of subject systems from the SF100 in the study (92).

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we described SUMO, an algorithm designed to solve the problems faced when using automatic remodular-isation algorithms. The interactive element of SUMO creates a virtuous cycle of improvement that is guaranteed to be con-sistent with the user's domain knowledge. Transitivity of positive relations enables SUMO to make the most of the information provided by the user, which we demonstrated in Theorem 2 as well as practically in a comprehensive sys-tematic evaluation.

The study, built on data collected from an observational pilot study, explored the performance of SUMO for a diverse set of software systems. The results indicate that SUMO produces accurate result from an amount of user input that falls well below the theoretical worst-case. They also indicate that in order to produce an approximate result, this can be achieved with an amount of input that is far lower than the amount required to obtain an exact result.Our ongoing and future work is focused on further improving SUMO. This involves the incorporation of auto-mated source code refactoring tools that work alongside SUMO [48], refining the constraint satisfaction algorithm to favour better results, and improving the user-interface to enable the user to be more expressive when specifying rela-tions between classes and modules. Alongside these improvements, we also intend to experiment with the appli-cation of SUMO in an industrial context.

The way in which we initialise the constraint solver leaves it free to select any solution, but more likely to choose similar solutions. This initialisation could be fur-ther improved by representing the similarity to the previ-ous solution explicitly by treating the problem as a constrained minimisation problem. This would require adoption of an appropriate objective function and is a strategy to the problem similar to Bavota et al.'s interactive GA [16]. We intend to evaluate to what extent we can improve the solutions produced by identifyCorrections without negatively impacting the speed at which it returns solutions. The proposed mIBEA introduces the ranking from Pareto dominance-based multi-objective evolutionary algorithm into the indicator-based algorithm during the elite solution preservation process.

The empirical results from mIBEA over the entire DTLZ benchmark functions show that mIBEA significantly improves the original IBEA on the coverage of resultant Pareto fronts, as well as hypervolume and +. Also, over 8fold speed-ups are obtained when using a larger population size. Also note that the proposed mIBEA does not introduce any additional parameter to the original IBEA.

REFERENCES

[1] T. Back, U. Hammel, and H.-P. Schwefel. Evolutionary computation: Comments on the history and current state. IEEE Transactions on Evolutionary Computation, 1(1): 3–17, 1997.

[2] N. Beume, B. Naujoks, and M. Emmerich. Sms-emoa: Multiobjective selection based on dominated

hypervolume. European Journal of Operational Research, 181 (3):1653–1669, 2007.

[3] K.Bringmann,T.Friedrich,F.Neumann,andM.Wagner. Approximation-guided evolutionary multi-objective optimization. In International Joint Conference on Artificial Intelligence (IJCAI), pages 1198–1203. AAAI, 2011.

[4] D. Brockhoff. A bug in the multiobjective optimizer ibea: Salutary lessons for code release and a performance reassessment. In Int. Conference on Evolutionary MultiCriterion Optimization, pages 187–201. Springer, 2015.

[5] S. Chand and M. Wagner. Evolutionary many-objectiv optimization: A quick-start guide. Surveys in Operations Research and Management Science, 20(2):35 – 42, 2015.

[6] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen, et al. Evolutionary algorithms for solving multi-objective problems, volume 5. Springer, 2007.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGAII. IEEE Transactions on Evolutionary Computation, 6 (2):182–197, 2002.

[8] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multiobjective optimization. Springer, 2005.

[9] F. E. C. Duran, C. Cotta, and A. J. Fern´andez-Leiva. A comparative study of multi-objective evolutionary algorithms to optimize the selection of investment portfolios with cardinality constraints. In European Conference on the Applications of Evolutionary Computation, pages 165–173. Springer, 2012.

[10] J. J. Durillo and A. J. Nebro. jmetal: A java framework

[11] B. S. Mitchell and S. Mancoridis, "On the automatic modulariza-tion of software systems using the bunch tool," IEEE Trans. Softw. Eng., vol. 32, no. 3, pp. 193–208, Mar. 2006.

[12] N. Anquetil and J. Laval, "Legacy software restructuring: Analyz-ing a concrete case," in Proc. 15th Eur. Conf. Softw. Maintenance Reengineering, 2011, pp. 279–286.

[13] R. W. Schwanke, "An intelligent tool for re-engineering software modularity," in Proc. 13th Int. Conf. Softw. Eng., 1991, pp. 83–92.

[14] M. Hall, N. Walkinshaw, and P. McMinn, "Supervised software modularisation," in Proc. 28th IEEE Int. Conf. Softw. Maintenance, 2012, pp. 472–481.

[15] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrodl,€ "Constrained k-means clustering with background knowledge," in Proc. 18th Int. Conf. Mach. Learn., 2001, pp. 577–584.

[16] G. Bavota, F. Carnevale, A. D. Lucia, M. D. Penta, and R. Oliveto, "Putting the developer in-the-loop: An interactive GA for soft-ware re-modularization," in Proc. 4th Int. Symp. Search Based Softw. Eng., 2012, pp. 75–89.

[17] L. Ponisio and O. Nierstrasz, "Using context information to re-architect a system," Proc. 3rd Softw. Meas. Eur. Forum, vol. 2006, pp. 91–103, 2006.

[18] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai, "Enhancing architectural recovery using concerns," in Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng., 2011, pp. 552–555.

[19] I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using cohesion and coupling for software remodularization: Is it enough?" ACM Trans. Softw. Eng. Methodology, vol. 25, no. 3, pp. 24:1–24:28, 2016. [Online]. Available: http://doi.acm.org/10.1145/2928268

[20] H. A. Muller,€ M. A. Orgun, S. R. Tilley, and J. S. Uhl, "A reverse-engineering approach to subsystem structure identification," J. Softw. Maintenance: Res. Practice, vol. 5, no. 4, pp. 181–204, Dec. 1993.

[21] B. Andreopoulos, A. An, V. Tzerpos, and X. Wang, "Multiple layer clustering of large software systems," in Proc. 12th Work. Conf. Reverse Eng., 2005, pp. 79–88.

[22] A. Marx, F. Beck, and S. Diehl, "Computer-aided extraction of software components," in Proc. 17th Work. Conf. Reverse Eng., 2010,PP183–192.