Finding Explanations for Multi-Objective Optimization (in Near-Linear Time)

Authors suppressed for blind review

Institution suppressed for blind review

Abstract. CT0 is an algorithm for summarizing trade-offs in multi-objective problems. From that summary, humans can read recommendations for their systems. This paper evaluates those recommendations using data generated from (a) the POM3 model of agile selection of tasks; (b) four COCOMO-suite predictors for software development effort, months, defects and risk.
CT0 is a very fast algorithm- both theoretically and empirically. For example, for the COCOMO-suite models, CT0 terminated in 3 seconds while standard optimizers (NSGA-II and SPEA2) took 150 seconds. Further, the generated explanations for CT0 were just as effective as from other optimizers.
Hence, we recommend CT0 when some succinct summary has to be rapidly generated (e.g. in some interactive design meeting). CT0 could also be useful as post-processor to other optimizers (to generate succinct explanations of their conclusions) or as a optimizer to other optimizers (by constraining those other optimizers to only search the regions recommended by CT0).

Keywords: Software engineering, explanation, optimization, multi-objective.

1 Introduction

"If you cannot- in the long run- tell everyone what you have been doing, your doing has been worthless."
– Erwin Schrödinger

Explaining the results of multi-objective optimization to a user can be problematic. A typical run of a multi-objective optimizer can process thousands to millions of examples. It is an overwhelming task for humans to certify the correctness of conclusions generated from so many results. Verrappa and Leiter warn that

> "..for industrial problems, these algorithms generate (many) solutions, which makes the tasks of understanding them and selecting one among them difficult and time consuming" [**?**].

Even if explanations are constrained to (say) just a few hundred examples taken from the Pareto frontier, this can still confuse the user. Valerdi notes that it can take days for panels of human experts to rigorously review even a few dozen examples [32]. For example, once had a client who disputed the results of our analysis. They demanded to audit the reasoning but when we delivered the of candidate solutions on the Pareto frontier, they were overwhelmed by the amount of information. Flustered, the client discounted the analysis and rejected our conclusions. From this experience, we learned that to better support decision making in SBSE, we must better explain SBSE results.

Other researchers have recognized the importance of explanation. It is known to be a key factor in selecting algorithms. For example, in the field of machine learning, "each time one of our favorite approaches has been applied in industry, each time the comprehensibility of the results, though ill-defined, has been a decisive factor of choice over an approach by pure statistical means, or by neural networks." [4]. Analogous terms to explainability in that community are "comprehensibility", "interpretability" [1] or "understandability" [3].

In spite of the importance attributed to the subject, explanation has not been extensively investigated in the context of SBSE. One of the few papers that does is that of Veerappa and Lieter [34] who clustered examples from the Pareto frontier (examples generated from a goal graph representation of requirements for London ambulance services). In this approach, "instead of having to inspect a large number of individual solutions, (users) can look at a much smaller number of groups of related solutions, and focus their attention on the important characteristics of the group rather than the particularities of their individual solutions" [34].

XXXX after creating. still errors in comparisons

While an innovative and insightful study, there are three open issues with that method: (a) the complexity of clustering; (b) erroneous conclusions could be generated from the users inspection of the clusters; (c) introduced by users incorrectly evaluation the value of generated recommendations. Veerappa and Lieter did not evaluate the effects of the recommendations that could be generated by users browsing their clusters. Also, their method could suffer from scalability issues since it a post-processor to a clustering algorithm (clustering is a slow process requiring say, $O(N^2)$ comparisons for the greedy agglomerate clustering algorithm used in that paper [20]).

Accordingly, in this paper, when:

1. Cluster using a near-linear time algorithm;
2. Better define the process by which recommendations are generated from clusters;
3. The generated recommendations are tested by generating more examples from the model *after* the recommendations are imposed as extra constraints on the model inputs.

2 A Motivating Example

SEI

3 Explaining "Explanation"

As a starting point in this exploration of explanation, it is important to distinguish between the (1) problem of explaining the output of an multi-objective optimizer (discussed in this paper); from the more complex problem of (2) explaining how that output was generated. To put that another way: we seek to explain eggs, but not the chicken.

Next, a definition of "explanation" is required such that:

1. An explanation system can be designed;
2. It is possible to distinguish a "good" for a "bad" explanation.

In the SE literature, the general consensus in software engineering is that "good" explanations are succinct explanations [4, 11, 12]. On this score, MOEAs fare poorly since their output can be very verbose (hundreds or more examples from the Pareto frontier).

Also, cognitive science theory argues that there is more to "explaining" something that just showing it succinctly. According to Kelly's personal construct theory (PCT) humans explain things via "constructs" that distinguish sets of examples [18]. So, for

Kelly, human explanations are not about "things" in isolation but rather the *differences between groups of things*. In data mining, finding differences between things is called *contrast set learning* [27].

Other cognitive science research studied the activities humans do during explanation generation. Leake [21] lists a dozen different tasks that humans perform when "explaining" some phenomena. Leake does not claim that the following list is complete; just that it demonstrates a wide range of goal-based purposes for explanation, including:

1. Connect event to expected/believed conditions.
2. Connect event to previously unexpected conditions.
3. Find predictors for anomalous situation.
4. Find repair points for causes of an undesirable state.
5. Clarify current situation to predict effects or choose response.
6. Find controllable (blockable or achievable) causes.
7. Find actors contributions to outcome.
8. Find motivations for anomalous actions or decisions.
9. Find a within-theory derivation.

That is, to Leake, explanation is akin to planning where the "explainer" is showing some audience how to find or connect together information. A system that supports such explanations makes it easier to "connect the dots". In practice that means an explanation system must:

- Input a large set of axioms: e.g. examples, pieces of background knowledge;
- Output a *reduced* set of axioms: e.g. rules, model fragments, or as done by Veerappa and Lieter [34], a small number of representative examples takes from centroids of clusters on the Pareto frontier;
- Such that, in the reduces space, it is simple and quick to generate goal-based explanations including the nine kinds listed above.

3.1 Decision Trees as "Explanation Tools"

Decision tree learning is a widely-used framework for data mining: given a single goal (called the "class"), find some attribute value that splits the data such that the distribution of classes in each split has been simplified (where the simplest distribution is one containing examples from only one class). Decision tree learners then grow sub-trees by recursing on the data in each split. Popular decision-tree learners include:

- CART [**?**, 6] which minimizes the variance of continuous classes in each split; or
- C4.5 [30] which minimizes the information content of the discrete classes in each split.

One reason to prefer decision trees is that they can very fast to execute. Each level of recursion processes progressively less data. Also, the computation at each level of the recursion may be just a few linear passes through the data, followed by an sort of the attributes– so nothing more than $O(Nlog(N))$ at each level [**?**].

Another reason to prefer decision trees is that, as discussed below, they can operationalize much of Leake's and Kelly's cognitive models on explanation. Decision tree learners do have the disadvantage in that, as used in standard pratice, they only focus on one goal. The aim of this paper is to present a novel extension to standard decision tree learning that extends them to multi-objective optimization.

Given the above discussion, it is easy to see why that is so since decision tree learners can operationalize the above definitions of "explanation".

One reason for the popularity of decision tree learners Previously, work on constrast learning for single goal SE problems found that very succinct contrast sets could be generated as a post-processor to decision tree learning [25]:

– Building a decision tree to separate the different outcomes;
– Identifying leaves containing desired outcome X and undesired outcome Y;
– Querying that tree to find branches B_x and B_y that lead to X, Y.
– Computing $B_x - B_y$ which selects/rejects for desired/undesired outcomes.

In one spectacularly successful demonstration of this technique [23], it was found decision trees with 6,000 nodes had much superfluous information. Specifically, when some branch point high in the tree most separated the classes, then all contrast set learning had to do is report those branch decisions that selected for branches leading to the better classes. Using that approach, a contrast set learning could report contrasts with only one to four variables in each (and when applied to test data, those constrast sets were at pruning away all the undesired outcomes). Other studies with other data sets [24] confirmed the **the law of tiny constrasts**: *the minimal constrast set between things is usually much smaller than a complete description of those things.*

For simple goal classification, one way to operationalize Leake's framework is using decision trees. Given leaves of that tree $\{X, Y, Z, etc\}$, then exists some branch $\{B_x, B_y, B_z, etc\}$ that connects the root to the leaves as a conjunction of attribute/value pairs. Given some opinion about the value of the contents of each leaf $\{U_x, U_Y, U_Z, etc\}$, then the set difference $B_x - B_y$ is the contrast set of the differences that can drive examples on X over to Y.

"from here to there".

an explanation does not generate some single unique output. Rather, it inputs a set of axioms or examples and outputs a reduced set of axioms or examples within which it faster and simpler to generate explanations

Current MOEA algorithms are "instance-based methods" that return specific examples that perform "best" with respect to the multiple goals. The number of examples generated in this way can be overwhelming.

If a user wants to learn general principles from those examples, some secondary *explanation* process is required to group and generalize those examples. For example, Veerappa and Lieter [34] clustering examples from the Pareto frontier so users (at a minimum) need only browse the centroids of each clusters).

GAs flat vectors, not the trees explored by by ()say) Gouse et al.

Goals is performance just as good but explain better

One caveat before beginning: if the audience for the results of optimization are not human beings, then perhaps an explanation systems is not required. For example, Petke, Harman, Langdon, & Weimer [29] use evolutionary methods to rewrite code such that the new code executes faster. The audience for the rewritten code is a compiler. Such compilers do not argue or and ask questions about the code they are given to process. Hence, that rewrite system does not necessary need an explanation system. That said, a succinct and useful description of the difference between passing and failing runs of the rewrite system could be useful when (e.g.) a human is trying to debug that code rewrite system.

Yet another model of "explanation" not explored here is the "surprise modeling" approach recommended by Freitas [13], Voinea&Tulea [2] and others including Horvitz [16]. In that approach, (a) some background knowledge (e.g. summaries of prior actions by users) is used to determine "normal" behavior; (b) users are only presented results that deviated from normal expectations. In analogous research, Koegh [19] argues that *time series discords* (infrequent sequential events in a times series) are a useful way to summarize reports from complex temporal streams. The premise of surprise modeling and reporting discords is that "rare events need to be explored". In non-temporal domains, time series discords becomes *anomaly detection* [7]. For example, in the SE domian, Voinea and Telea report tools that can quickly highlight regions of unusually active debugging (and such regions should be reviewed by management) [**?**] (see also the anomaly detection work of Gruska et al. [14]).

We do not dispute the importance of exploring anomalous outliers. On the other hand, when forming policies for software projects, we need treatments that are well supported by the data. Hence, our contrast sets report changes in the data that, in our data, were *frequently* seen to lead to change.

Also, time series discords and anomaly detection are reports on some variables. Hence, they have a different goal to CT0 that strives to report recommendations on how to change the system so to remove some problem.

Further, all the systems described above [2, 14, 16, 19] are either for unsupervised learning (where no objectives are known) or for single objective systems (where only one goal is known). CT0, on the other hand, is more ambitious since it was designed for multi-objective systems.

Another potential issue with CT0 is correlation-vs-causation conflation. The issue here is that contrast sets will be useless if they report spurious correlations and not true causal effects. Proving that some effect is truly causal is a non-trivial task. The standard Hall criteria for causal effects [28] is so strict that, outside of highly controlled lab conditions, it rarely accepts that any effect is causal. Hence, in software engineering, when researchers talk of causality [5, 10, 17, 35] they use Granger's "predictive causality"; i.e. causality is the ability of predicting values seen in the future from values seen in the past. Elsewhere, Granger causality has been adapted to data mining by organizing cross-validations such that the test sets contain data collected at a later time than the training sets [22]. In this paper, we adapt Granger casualty to search-based methods by testing recommendations learned from M simulations on a subsequent round of N new simulations. Those recommendations satisfy Granger causality when the subsequent round of N simulations are changed in a manner predicted by the recommendations gleaned from the original M simulations.

"Data farming" is a technique used extensively by the U.S. Military [**?**]. Data farming builds a "landscape" of output that can be analyzed for trends, anomalies, and insights in multiple parameter dimensions. In a recent review of search-based and data mining methids in SE, we found numerous examples of data farming [**?**, **?**, **?**, **?**, 8, 9, 15, 26, 31, 33].

In theory. Once a project manager can view their project on the landscape, they can use this visualization to determine

We come to this work after attending a recent seminar at the US Department of Defence's Software Engineering Institute (SEI), Pittsburgh, USA. That seminar reflected on how to best broadcast the lessons learned by SEI to a very broad audience.

In the 21^{st} century, it is now impossible to manually browse very large quantities of software project data. For example, as of October 2012, Mozilla Firefox had 800K reports on software projects. While it is now possible to automatically analyze such data with data miners, at some stage a group of business users will have to convene to *interpret the results* (e.g., to decide if it is wise to deploy the results as a defect reduction method within an organization). These business users are now demanding that data mining tools be augmented with tools to support business-level interpretation of that data. For example,

at a recent panel on software analytics at ICSE'12,
industrial practitioners lamented the state of the art in data mining
and software engineering [**?**]. Panelists commented that
"prediction is all well and good, but what about decision
making?". That is, these panelists are more interested in the interpretations
that follow the mining, rather than just the mining.

4 Related Work

Sayyad

GALE

References

1.
2. Visual data mining and analysis of software repositories, 2007.
3. H. Allahyari and N. Lavesson. User-oriented assessment of classification model understandability. In *SCAI'11*, pages 11–19, 2011.
4. I. Askira-Gelman. Knowledge discovery: Comprehensibility of the results. In *Hawaii International Conference on System Sciences*, 1998.
5. P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos. Graph-based analysis and prediction for software evolution. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 419–429, Piscataway, NJ, USA, 2012. IEEE Press.
6. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth International, Monterey, CA, 1984.
7. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.
8. E. Chiang and T. Menzies. Simulations for very early lifecycle quality evaluations. *Software Process: Improvement and Practice*, 7(3-4):141–159, 2003. Available from `http://menzies.us/pdf/03spip.pdf`.
9. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
10. C. Couto, M. Valente, P. Pires, A. Hora, N. Anquetil, and R. Bigonha. Bugmaps-granger: a tool for visualizing and predicting bugs using granger causality tests. *Journal of Software Engineering Research and Development*, 2, 1024.
11. K. Dejaeger, T. Verbraken, and B. Baesens. Toward comprehensible software fault prediction models using bayesian network classifiers. *Software Engineering, IEEE Transactions on*, 39(2):237–257, Feb 2013.
12. N. E. Fenton and M. Neil. Software metrics: Success, failures and new directions. *J. Syst. Softw.*, 47(2-3):149–157, July 1999.
13. A. A. Freitas. On objective measures of rule surprisingness. In *Proceedings of the Second European Conference on the Principles of Data Mining and Knowledge Discovery (PKDD'98*, pages 1–9. Springer-Verlag, 1998.
14. N. Gruska, A. Wasylkowski, and A. Zeller. Learning from 6,000 projects: lightweight cross-project anomaly detection. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 119–130. ACM, 2010.
15. W. Heaven and E. Letier. Simulating and optimising design decisions in quantitative goal models. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 79–88, 2011.
16. E. Horvitz, J. Apacible, R. Sarin, and L. Liao. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *UAI'05*, pages 275–283, 2005.

17. B. Huberman, D. Romero, and F. Wu. Crowdsourcing, attention and productivity. *Journal of Information Science*, 35(6):758–765, December 2009.
18. G. Kelly. *The Psychology of Persona] Constructs. Volume 1: A Theory of Personality. Volume 2: Clinical Diagnosis and Psychotherapy*. Norton, 1955.
19. E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 226–233, Washington, DC, USA, 2005. IEEE Computer Society.
20. E. Kocaguneli, T. Menzies, A. Bener, and J. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 28:425–438, 2012. Available from `http://menzies.us/pdf/11teak.pdf`.
21. D. Leake. Goal-based explanation evaluation. *Cognitive Science*, 15:509–545, 1991.
22. M. Lumpe, R. Vasa, T. Menzies, R. Rush, and R. Turhan. Learning better inspection optimization policies. *International Journal of Software Engineering and Knowledge Engineering*, 21(45):725–753, 2011.
23. T. Menzies and Y. Hu. Data mining for very busy people. November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.
24. T. Menzies and Y. Hu. Just enough learning (of association rules): The TAR2 treatment learner. In *Artificial Intelligence Review*, 2007. Available from `http://menzies.us/pdf/07tar2.pdf`.
25. T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://menzies.us/pdf/00ase.pdf`.
26. I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Trans. Softw. Eng.*, 31(5):380–391, May 2005.
27. P. K. Novak, N. Lavrač, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *J. Mach. Learn. Res.*, 10:377–403, June 2009.
28. L. Paul and N. Hall. *Causation : A Users Guide*. Oxford University Press, 2013.
29. J. Petke, M. Harman, W. Langdon, and W. Weimer. Using genetic improvement & code transplants to specialise a c++ program to a problem class. In *European Conference on Genetic Programming (EuroGP)*, 2014.
30. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.
31. M. Shepperd and G. F. Kadoda. Comparing software prediction techniques using simulation. *IEEE Trans. Software Eng*, 27(11):1014–1022, 2001.
32. R. Valerdi. Convergence of expert opinion via the wideband delphi method: An application in cost estimation models. In *Incose International Symposium, Denver, USA*, 2011. Available from http://goo.gl/Zo9HT.
33. A. van Lamsweerde and E. Letier. Integrating obstacles in goal-driven requirements engineering. In *Proceedings of the 20th International Conference on Software Engineering*, pages 53–62. IEEE Computer Society Press, 1998. Available from `http://citeseer.nj.nec.com/vanlamsweerde98integrating.html`.
34. V. Veerappa and E. Letier. Understanding clusters of optimal solutions in multi-objective decision problems. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*, RE '11, pages 89–98, Washington, DC, USA, 2011. IEEE Computer Society.
35. P. Zheng, Y. Zhou, M. Lyu, and Y. Qi. Granger causality-aware prediction and diagnosis of software degradation. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 528–535, June 2014.