Noname manuscript No.
(will be inserted by the editor)

Hyperparameter Optimization for Effort Estimation

**Tianpei Xia · Rahul Krishna · Jianfeng Chen ·
George Mathew · Xipeng Shen · Tim Menzies**

the date of receipt and acceptance should be inserted later

Abstract Software analytics has been widely used in software engineering for many tasks. One of the "black arts" of software analytics is tuning the parameters controlling a machine learning algorithm. Such a *hyperparameter optimization* has been found to be very useful in SE defect prediction and text mining. [1, 25, 24, 56, 7] Accordingly, this paper seeks simple, automatic, effective and fast methods for finding tunings for software effort estimation.

We introduce a hyperparameter optimization architecture called OIL (Optimized Inductive Learning). OIL is tested on a wide range of optimizers using data from 945 projects. After tuning, large improvements in effort estimation accuracy were observed (measured in terms of magnitude of relative error and standardized accuracy).

From those results, we recommend using regression trees (CART) tuned by either different evolution or FLASH (a sequential model optimizer). This particular combination of learner and optimizers achieves superior results in a few minutes (rather than the many days required by some other approaches).

Keywords Effort Estimation · Hyperparameter Optimization · Regression Trees · Analogy

1 Introduction

Software analytics has been widely used in software engineering for many tasks [52]. This paper explores methods to improve algorithms for software effort estimation (a particular kind of analytics tasks). This is needed since software effort estimates can be wildly inaccurate [34]. Effort estimations need to be accurate (if for no other reason) since many government organizations demand that the budgets allocated to large publicly funded projects be double-checked by some estimation model [55].

Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, Tim Menzies
Department of Computer Science, North Carolina State University, Raleigh, NC, USA
Email: txia4@ncsu.edu, rkrish11@ncsu.edu, jchen37@ncsu.edu, george2@ncsu.edu, xshen5@ncsu.edu, timm@ieee.org

Non-algorithm techniques that rely on human judgment [30] are much harder to audit or dispute (e.g., when the estimate is generated by a senior colleague but disputed by others).

Sarro et al. [73] assert that effort estimation is a critical activity for planning and monitoring software project development in order to deliver the product on time and within budget [12, 39, 85]. The competitiveness of software organizations depends on their ability to accurately predict the effort required for developing software systems; both over- or under- estimates can negatively affect the outcome of software projects [85, 49, 50, 79].

Hyperparameter optimizers tuning the control parameters of a data mining algorithm. It is well established that classification tasks like software defect prediction or text classification are improved by such tuning [24, 84, 2, 1]. This paper investigates hyperparameter optimization using data from 945 projects. To the best of our knowledge, this study is an extensive exploration of hyperparameter optimization and effort estimation following the earlier work by [17].

We assess our results with respect to recent findings by Arcuri & Fraser [6]. They caution that to transition hyperparameter optimizers to industry, they need to be **fast**:

> *A practitioner, that wants to use such tools, should not be required to run large tuning phases before being able to apply those tools [6].*

Also, according to Arcuri & Fraser, optimizers must be **useful**:

> *At least in the context of test data generation, (tuning) does not seem easy to find good settings that significantly outperform "default" values. ... Using "default" values is a reasonable and justified choice, whereas parameter tuning is a long and expensive process that might or might not pay off [6].*

Hence, to assess such optimization for effort estimation, we ask four questions.

RQ1: To address one concern raised by Arcuri & Fraser, we must first ask *is it best to just use "off-the-shelf" defaults?* We will find that tuned learners provide better estimates than untuned learners. Hence, for effort estimation:

Lesson1: "off-the-shelf" defaults should be deprecated.

RQ2: Can tuning effort be avoided by *replacing old defaults with new defaults*? This checks if we can run tuning once (and once only) then use those new defaults ever after. We will observe that effort estimation tunings differ extensively from dataset to dataset. Hence, for effort estimation:

Lesson2: Overall, there are no "best" default settings.

RQ3: The first two research questions tell us that we must retune our effort estimators whenever new data arrives. Accordingly, we must now address the other concern raised by Arcuri & Fraser about CPU cost. Hence, in this question we ask *can we avoid slow hyperparameter optimization*? The answer to **RQ3** will be "yes" since our results show that for effort estimation:

Lesson3: Overall, our slowest optimizers perform no better than faster ones.

RQ4: The final question to answer is *what hyperparameter optimizers to use for effort estimation?* Here, we report that a certain combination of learners and optimizers usually produce best results. Further, this particular combination often achieves in a few minutes what other optimizers may need hours to days of CPU to achieve. Hence we will recommend the following combination for effort estimation:

Lesson4: For new datasets, try a combination of *CART* with the optimizers *differential evolution* and *FLASH*.

(Note: The *italicized* words are explained below.)

In summary, unlike the test case generation domains explored by Arcuri & Fraser, hyperparamter optimization for effort estimation is both **useful** and **fast**.

Overall the contributions of this paper are:

- A demonstration that defaults settings are not the best way to perform effort estimation. Hence, when new data is encountered, some tuning process is required to learn the best settings for generating estimates from that data.
- A recognition of the inherent difficulty associated with effort estimation. Since there is not one universally best effort estimation method. commisioning a new effort estimator requires extensive testing. As shown below, this can take hours to days of CPU time.
- The identification of a combination of learner and optimizer that works as well as anything else, and which takes minutes to learn an effort estimator.
- An extensible open-source architecture called OIL that enables the commissioning of effort estimation methods. OIL makes our results repeatable and refutable.

The rest of this paper is structured as follows. The next section discusses different methods for effort estimation and how to optimize the parameters of effort estimation methods. This is followed by a description of our data, our experimental methods, and our results. After that, a discussion section explores open issues with this work.

From all of the above, we can conclude that (a) Arcuri & Fraser's pessimism about hyperparameter optimization applies to their test data generation domain. However (b) for effort estimation, hyperparamter optimization is both **useful** and **fast**. Hence, we hope that OIL, and the results of this paper, will prompt and enable more research on methods to tune software effort estimators.

Note that OIL and all the data used in this study is freely available for download from https://github.com/ai-se/magic101

2 About Effort Estimation

Software effort estimation is a method to offer managers approximate advice on how much human effort (usually expressed in terms of hours, days or months of human work) is required to plan, design and develop a software project. Such advice can only ever be approximate due to dynamic nature of any software development. Nevertheless, it is important to attempt to allocate resources properly in software projects to avoid waste. In some cases, inadequate or overfull funding can cause a considerable waste of resource and time. For example, NASA canceled its Check-out Launch Control System project after the initial \$200M estimate was exceeded by another \$200M [18]. As shown below, effort estimation can be categorized into (a) human-based and (b) algorithm-based methods [40, 74].

For several reasons, this paper does not explore human-based estimation methods. Firstly, it is known that humans rarely update their human-based estimation knowledge based on feedback from new projects [32]. Secondly, algorithm-based methods are preferred when estimate have to be audited or debated (since the method is explicit and available for inspection). Thirdly, algorithm-based methods can be run many times (each time applying small mutations to the input data) to understand the range of possible estimates. Even very strong advocates of human-based methods [31] acknowledge that algorithm-based methods are useful for learning the uncertainty about particular estimates.

2.1 Algorithm-based Methods

There are many algorithmic estimation methods. Some, such as COCOMO [10], make assumptions about the attributes in the model. For example, COCOMO requires that data includes 22 specific attributes such as analyst capability (acap) and software complexity (cplx). This attribute assumptions restricts how much data is available for studies like this paper. For example, here we explore 945 projects expressed using a wide range of attributes. If we used COCOMO, we could only have accessed an order of magnitude fewer projects.

Due to its attribute assumptions, this paper does not study COCOMO data. All the following learners can accept projects described using any attributes, just as long as one of those is some measure of project development effort.

Whigham et al.'s ATLM method [87] is a multiple linear regression model which calculate the effort as $\mathit{effort} = \beta_0 + \sum_i \beta_i \times a_i + \varepsilon_i$, where a_i are explanatory attributes and ε_i are errors to the actual value. The prediction weights β_i are determined using least square error estimation [65]. Additionally, transformations are applied on the attributes to further minimize the error in the model. In case of categorical attributes the standard approach of "dummy variables" [28] is applied. While, for continuous attributes, transformations such as logarithmic, square root, or no transformation is employed such that the skewness of the attribute is minimum. It should be noted that, ATLM does not consider relatively complex techniques like using model residuals, box transformations or step-wise regression (which are standard) when developing a linear regression model. The authors make this decision since they intend ATLM to be a simple baseline model rather than the "best" model. One aim of ATLM is that apart from it being simple and so requiring no decisions re parameters from the user, is that it can be applied automatically - and as such there should be no excuse not to compare any new model against a comparatively naive baseline.

Sarro et al. proposed a method named Linear Programming for Effort Estimation (LP4EE) [72], which aims to achieve the best outcome from a mathematical model with a linear objective function subject to linear equality and inequality constraints.

Table 1 CART's parameters.

Parameter	Default	Tuning Range	Notes
max_feature	None	[0.01, 1]	The number of feature to consider when looking for the best split.
max_depth	None	[1, 12]	The maximum depth of the tree.
min_sample_split	2	[0, 20]	Minimum samples required to split internal nodes.
min_samples_leaf	1	[1, 12]	Minimum samples required to be at a leaf node.

The feasible region is given by the intersection of the constraints and the Simplex (linear programming algorithm) is able to find a point in the polyhedron where the function has the smallest error in polynomial time. In effort estimation problem, this model minimises the Sum of Absolute Residual (SAR), when a new project is presented to the model, LP4EE predicts the effort as $effort = a_1 * x_1 + a_2 * x_2 + ... + a_n * x_n$, where x is the value of given project feature and a is the corresponding coefficient evaluated by linear programming. LP4EE is suggested to be used as another baseline model for effort estimation since it provides similar or more accurate estimates than ATLM and is much less sensitive than ATLM to multiple data splits and different cross-validation methods.

Some algorithm-based estimators are regression trees such as CART [46], CART is a tree learner that divides a dataset, then recurses on each split. If data contains more than *min_sample_split*, then a split is attempted. On the other hand, if a split contains no more than *min_samples_leaf*, then the recursion stops. CART finds the attributes whose ranges contain rows with least variance in the number of defects. If an attribute ranges r_i is found in n_i rows each with an effort variance of v_i, then CART seeks the attribute with a split that most minimizes $\sum_i \left(\sqrt{v_i} \times n_i / (\sum_i n_i)\right)$. For more details on the CART parameters, see Table 1.

Random Forest [11] and Support Vector Regression [14] are another instances of regression methods. Random Forest (RF) is an ensemble learning method for regression (and classification) tasks that builds a set of trees when training the model. To decide the output, it uses the mode of the classes (classification) or mean prediction (regression) of the individual trees. Support Vector Regression (SVR) uses kernel functions to project the data onto a new hyperspace where complex non-linear patterns can be simply represented. It aims to construct an optimal hyperplane that fits data and predicts with minimal empirical risk and complexity of the modelling function.

Another algorithm-based estimators are the analogy-based Estimation (ABE) methods advocted by Shepperd and Schofield [76]. ABE is widely-used [68, 42, 29, 38, 55], in many forms. We say that "ABE0" is the standard form seen in the literature and "ABEN" are the 6,000+ variants of ABE defined below. The general form of ABE (which applies to ABE0 or ABEN) is to first form a table of rows of past projects. The *columns* of this table are composed of independent variables (the *features* that define projects) and one dependent *feature* (project effort). From this table, we learn what similar projects (analogies) to use from the training set when examining a new test instance. For each test instance, ABE then selects k analogies out of the training set. Analogies are selected via a *similarity measure*. Before calculating similarity, ABE normalizes numerics min..max to 0..1 (so all numerics get equal chance to influence

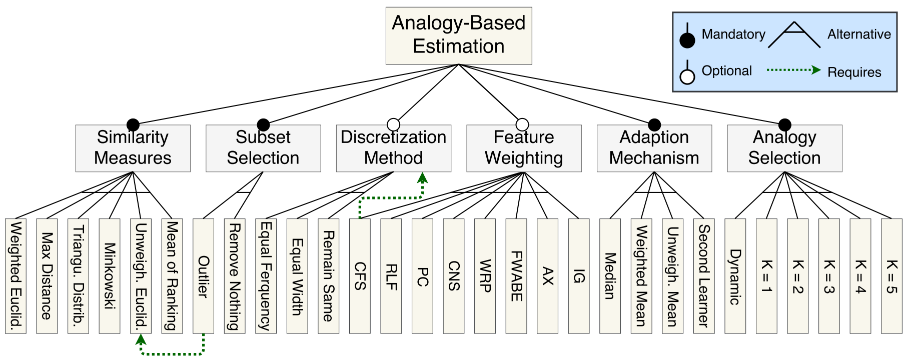


Fig. 1 OIL's feature model of the space of machine learning options for ABEN. In this model, $SubsetSelection$, $Similarity$, $AdaptionMechanism$ and $AnalogySelection$ are the mandatory *features*, while the $FeatureWeighting$ and $DiscretizationMethod$ *features* are optional. To avoid making the graph too complex, some cross-tree constrains are not presented. For more details on the terminology of this figure, see Table 2.

the dependent). Then, ABE uses *feature* weighting to reduce the influence of less informative *features*. Finally, some *adaption strategy* is applied return a combination of the dependent effort values seen in the k nearest analogies. For details on ABE0 and ABEN, see Figure 1 & Table 2.

2.2 Effort Estimation and Hyperparameter Optimization

Note that we do *not* claim that the above represents all methods for effort estimation. Rather, we say that (a) all the above are either prominent in the literature or widely used; and (b) anyone with knowledge of the current effort estimation literature would be tempted to try some of the above.

Even though our lost of effort estimation methods is incomplete, it is still very long. Consider, for example, just the ABEN variants documented in Table 2. There are $2 \times 8 \times 3 \times 6 \times 4 \times 6 = 6,912$ such variants. Some can be ignored; e.g. at $k = 1$, adaptation mechanisms return the same result, so they are not necessary. Also, not all *feature* weighting techniques use discretization. But even after those discards, there are still thousands of possibilities.

Given the space to exploration is so large, some researchers have offered automatic support for that exploration. Some of that prior work suffered from being applied to limited data [48].

Other researchers assume that the effort model is a specific parametric form (e.g. the COCOMO equation) and propose mutation methods to adjust the parameters of that equation [3, 58, 78, 13, 71]. As mentioned above, this approach is hard to test since there are very few datasets using the pre-specified COCOMO attributes.

Further, all that prior work needs to be revisited given the existence of recent and very prominent methods; i.e. ATLM from TOSEM'2015 [87] or LP4EE from TOSEM'2018 [72].

Table 2 Variations on analogy. Visualized in Figure 1.

– To measure similarity between x, y, ABE uses $\sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}$ where w_i corresponds to *feature* weights applied to independent *features*. ABE0 uses a uniform weighting where $w_i = 1$. ABE0's *adaptation strategy* is to return the effort of the nearest $k = 1$ item.
– *Two ways to find training subsets*: (a) Remove nothing: Usually, effort estimators use all training projects [15]. Our ABE0 is using this variant; (b) Outlier methods: prune training projects with (say) suspiciously large values [36].
– *Eight ways to make feature weighting*: Li *et al.* [48] and Hall and Holmes [27] review 8 different *feature* weighting schemes.
– *Three ways to discretize* (summarize numeric ranges into a few bins): Some *feature* weighting schemes require an initial discretization of continuous columns. There are many discretization policies in the literature, including: (1) equal frequency, (2) equal width, (3) do nothing.
– *Six ways to choose similarity measurements:* Mendes *et al.* [51] discuss three similarity measures, including the weighted Euclidean measure described above, an unweighted variant (where w_i = 1), and a "maximum distance" measure that focuses on the single *feature* that maximizes interproject distance. Frank *et al.* [23] use a triangular distribution that sets to the weight to zero after the distance is more than "k" neighbors away from the test instance. A fifth and sixth similarity measure are the Minkowski distance measure used in [4] and the mean value of the ranking of each project *feature* used in [86].
– *Four ways for adaption mechanisms:* (1) median effort value, (2) mean dependent value, (3) summarize the adaptations via a second learner (e.g., linear regression) [48, 53, 8, 70], (4) weighted mean [51].
– *Six ways to select analogies:* Analogy selectors are fixed or dynamic [40]. Fixed methods use $k \in \{1, 2, 3, 4, 5\}$ nearest neighbors while dynamic methods use the training set to find which $1 \leq k \leq N - 1$ is best for N examples.

Accordingly, this paper conducts a more thorough investigation of hyperparameter optimization for effort estimation.

– We use methods with no data feature assumptions (i.e. no COCOMO data);
– That vary many parameters (6,000+ combinations);
– That also tests results on 9 different sources with data on 945 software projects;
– Which uses optimizers representative of the state-of-the-art (Differential Evolution [81], FLASH [63]);
– And which benchmark results against prominent methods such as ATLM and LP4EE.

2.3 OIL

OIL is our architecture for exploring hyperparameter optimization and effort estimation, initially, our plan was to use standard hyperparameter tuning for this task. Then we learned that standard machine learning toolkits like Scikit-learn [67] did not include many of the effort estimation techniques; and (b) standard hyperparameter tuners can be slow. Hence, we build OIL:

– At the base *library layer*, we use Scikit-learn [67].
– Above that, OIL has a *utilities layer* containing all the algorithms missing in Scikit-Learn (e.g., ABEN required numerous additions at the utilities layer).

- Higher up, OIL's *modelling layer* uses an XML-based domain-specific language to specify a *feature* map of predicting model options. These feature models are single-parent and-or graphs with (optional) cross-tree constraints showing what options require or exclude other options. A graphical representation of the feature model used in this paper is shown in Figure 1.
- Finally, at top-most *optimizer layer*, there is some optimizer that makes decisions across the *feature* map. An automatic *mapper* facility then links those decisions down to the lower layers to run the selected algorithms.

2.4 Optimizers

Once OIL's layers were built, it was simple to "pop the top" and replace the top layer with another optimizer. Nair et al. [62] advise that for search-based SE studies, optimizers should be selecting via the a "dumb+two+next" rule. Here:

- "Dumb" are some baseline methods;
- "Two" are some well-established optimizers;
- "Next" are more recent methods which may not have been applied before to this domain.

For our "dumb" optimizer, we used Random Choice (hereafter, RD). To find N valid configurations, RD selects leaves at random from Figure 1. All these N variants are executed and the best one is selected for application to the test set. To maintain a fair comparison with other systems described below, OIL Chooses N as the same number of evaluations in other methods.

Moving on, our "two" well-established optimizers are ATLM [87] and LP4EE [72]. For LP4EE, we perform experiments with the open source code provided by orginal authors. For ATLM, since there is no online source code available, we carefully re-implemented the method by ourselves.

As to our "next" optimizers, we used Differential Evolution (hereafter, DE [81]) and FLASH [63]. The premise of DE is that the best way to mutate the existing tunings is to extrapolate between current solutions. Three solutions a, b, c are selected at random. For each tuning parameter k, at some probability cr, we replace the old tuning x_k with y_k. For booleans $y_k = \neg x_k$ and for numerics, $y_k = a_k + f \times (b_k - c_k)$ where f is a parameter controlling differential weight. The main loop of DE runs over the population of size np, replacing old items with new candidates (if new candidate is better). This means that, as the loop progresses, the population is full of increasingly more valuable solutions (which, in turn, helps extrapolation). As to the control parameters of DE, using advice from Storn [81], we set $\{np, g, cr\} = \{20, 0.75, 0.3\}$. The number of generations gen was set to 10 to test the effects of a very CPU-light effort estimator.

FLASH, proposed by Nair e. al. [63], is an incremental optimizer. Previously, it has been applied to configuration system parameters for software systems. This paper is the first application of FLASH to effort estimation. Formally, FLASH is a *sequential model-Based optimizer* [**?**] (also known in the machine learning literature as an *active learner* [19] or, in the statistics literature as *optimal experimental design* [66]).

Whatever the name, the intuition is the same: reflects on the model built to date in order to find the next best example to evaluate. To tune a data miner, FLASH explores N possible tunings as follows:

1. Set the evaluation budget b. In order to make a fair comparison with other methods, we used $b = 200$.
2. Run the data miner using $n = 20$ randomly selected tunings.
3. Build an *archive* of n examples holding pairs of parameter settings and their resulting performance scores (e.g. MRE, SA, etc).
4. Using that archive, learn a *surrogate* to predicts performance. Following the methods of Nair et al. [63], we used CART [46] for that surrogate.
5. Use the surrogate to guess M performance scores where $M < N$ and $M \gg n$ parameter settings. Note that this step is very fast since it all that is required is to run M vectors down some very small CART trees.
6. Using some *selection function*, select the most "interesting" setting. After Nair et al. [63] we returned the setting with the worst prediction (i.e. find the most troubling possibility).
7. Collect performance scores by evaluating "interesting" using the data miners (i.e. check the most troubling possibility). Set $b = b - 1$.
8. Add "interesting" to the archive. If $b > 0$, goto step 4. Else, halt.

One way to summary the above is as follows. Given what we already know about the tunings (represented in a CART tree), find the potentially weirdest thing (in Step6), check that weird thing (in Step7) and update the model with that result. Repeat.

3 Empirical Study

3.1 Data

To assess OIL, we applied it to the 945 projects seen in nine datasets from the SEACRAFT repository[1]; see Table 3 and Table 4. This data was selected since it has been widely used in previous estimation research. Also, it is quite diverse since it differs in number of observations (from 15 to 499 projects); geographical locations (software projects coming from Canada, China, Finland); technical characteristics (software projects developed in different programming languages and for different application domains, ranging from telecommunications to commercial information systems); and number and type of features (from

Table 3 Data used in this study. For details on the features, see Table 4.

	Projects	Features
kemerer	15	6
albrecht	24	7
isbsg10	37	11
finnish	38	7
miyazaki	48	7
maxwell	62	25
desharnais	77	6
kitchenham	145	6
china	499	16
total	945	

[1] http://tiny.cc/seacraft

Table 4 Descriptive Statistics of the Datasets. Terms in *italics* are removed from this study, for reasons discussed in the text.

	feature	min	max	mean	std
kemerer	*Duration*	5	31	14.3	7.5
	KSLOC	39	450	186.6	136.8
	AdjFP	100	2307	999.1	589.6
	RAWFP	97	2284	993.9	597.4
	Effort	23	1107	219.2	263.1
albrecht	Input	7	193	40.2	36.9
	Output	12	150	47.2	35.2
	Inquiry	0	75	16.9	19.3
	File	3	60	17.4	15.5
	FPAdj	1	1	1.0	0.1
	RawFPs	190	1902	638.5	452.7
	AdjFP	199	1902	647.6	488.0
	Effort	0	105	21.9	28.4
isbsg10	UFP	1	2	1.2	0.4
	IS	1	10	3.2	3.0
	DP	1	5	2.6	1.1
	LT	1	3	1.6	0.8
	PPL	1	14	5.1	4.1
	CA	1	2	1.1	0.3
	FS	44	1371	343.8	304.2
	RS	1	4	1.7	0.9
	FPS	1	5	3.5	0.7
	Effort	87	14453	2959	3518
finnish	hw	1	3	1.3	0.6
	at	1	5	2.2	1.5
	FP	65	1814	763.6	510.8
	co	2	10	6.3	2.7
	prod	1	29	10.1	7.1
	lnsize	4	8	6.4	0.8
	lneff	6	10	8.4	1.2
	Effort	460	26670	7678	7135
china	*AFP*	9	17518	486.9	1059
	Input	0	9404	167.1	486.3
	Output	0	2455	113.6	221.3
	Enquiry	0	952	61.6	105.4
	File	0	2955	91.2	210.3
	Interface	0	1572	24.2	85.0
	Added	0	13580	360.4	829.8
	changed	0	5193	85.1	290.9
	Deleted	0	2657	12.4	124.2
	PDR_A	0	84	11.8	12.1
	PDR_U	0	97	12.1	12.8
	NPDR_A	0	101	13.3	14.0
	NPDU_U	0	108	13.6	14.8
	Resource	1	4	1.5	0.8
	Dev.Type	0	0	0.0	0.0
	Duration	1	84	8.7	7.3
	Effort	26	54620	3921	6481

	feature	min	max	mean	std
miyazaki	*KLOC*	7	390	63.4	71.9
	SCRN	0	150	28.4	30.4
	FORM	0	76	20.9	18.1
	FILE	2	100	27.7	20.4
	ESCRN	0	2113	473.0	514.3
	EFORM	0	1566	447.1	389.6
	EFILE	57	3800	936.6	709.4
	Effort	6	340	55.6	60.1
maxwell	App	1	5	2.4	1.0
	Har	1	5	2.6	1.0
	Dba	0	4	1.0	0.4
	Ifc	1	2	1.9	0.2
	Source	1	2	1.9	0.3
	Telon.	0	1	0.2	0.4
	Nlan	1	4	2.5	1.0
	T01	1	5	3.0	1.0
	T02	1	5	3.0	0.7
	T03	2	5	3.0	0.9
	T04	2	5	3.2	0.7
	T05	1	5	3.0	0.7
	T06	1	4	2.9	0.7
	T07	1	5	3.2	0.9
	T08	2	5	3.8	1.0
	T09	2	5	4.1	0.7
	T10	2	5	3.6	0.9
	T11	2	5	3.4	1.0
	T12	2	5	3.8	0.7
	T13	1	5	3.1	1.0
	T14	1	5	3.3	1.0
	T15	1	5	3.3	0.7
	Dura.	4	54	17.2	10.7
	Size	48	3643	673.3	784.1
	Time	1	9	5.6	2.1
	Effort	583	63694	8223	10500
desharnais	TeamExp	0	4	2.3	1.3
	MngExp	0	7	2.6	1.5
	Length	1	36	11.3	6.8
	Trans.s	9	886	177.5	146.1
	Entities	7	387	120.5	86.1
	AdjPts	73	1127	298.0	182.3
	Effort	546	23940	4834	4188
kitchenham	code	1	6	2.1	0.9
	type	0	6	2.4	0.9
	duration	37	946	206.4	134.1
	fun_pts	15	18137	527.7	1522
	estimate	121	79870	2856	6789
	esti_mtd	1	5	2.5	0.9
	Effort	219	113930	3113	9598

6 to 25 features, including a variety of features describing the software projects, such as number of developers involved in the project and their experience, technologies used, size in terms of Function Points, etc.);

Note that some features of the original datasets are not used in our experiment because they are (1) naturally irrelevant to their effort values (e.g., ID, Syear), (2) unavailable at the prediction phase (e.g., duration, LOC), (3) highly correlated or overlap to each other (e.g., raw function point and adjusted function points). A data cleaning process is applied to solve this issue. Those removed features are shown as italic in Table 4.

3.2 Cross-Validation

Each datasets was treated in a variety of ways. Each *treatment* is an *M*N-way* cross-validation test of some learner or some learner and optimizer. That is, M times, shuffle the data randomly (using a different random number seed) then divide the data into

Table 5 Performance scores: MRE and SA

MRE: MRE is defined in terms of AR, the magnitude of the absolute residual. This is computed from the difference between predicted and actual effort values:

$$AR = |actual_i - predicted_i|$$

MRE is the magnitude of the relative error calculated by expressing AR as a ratio of actual effort:

$$MRE = \frac{|actual_i - predicted_i|}{actual_i}$$

MRE has been criticized [22, 37, 43, 69, 77, 80] as being biased towards error underestimations. Nevertheless, we use it here since there exists known baselines for human performance in effort estimation, expressed in terms of MRE [59]. The same can not be said for SA.

SA: Because of issues with MRE, some researchers prefer the use of other (more standardized) measures, such as Standardized Accuracy (SA) [45, 75]. SA is defined in terms of

$$MAE = \frac{1}{N}\sum_{i=1}^{n}|RE_i - EE_i|$$

where N is the number of projects used for evaluating the performance, and RE_i and EE_i are the actual and estimated effort, respectively, for the project i. SA uses MAE as follows:

$$SA = (1 - \frac{MAE_{P_j}}{MAE_{r_{guess}}}) \times 100$$

where MAE_{P_j} is the MAE of the approach P_j being evaluated and $MAE_{r_{guess}}$ is the MAE of a large number (e.g., 1000 runs) of random guesses. Over many runs, $MAE_{r_{guess}}$ will converge on simply using the sample mean [75]. That is, SA represents how much better P_j is than random guessing. Values near zero means that the prediction model P_j is practically useless, performing little better than random guesses [75].

N bins. For $i \in N$, bin i is used to test a model build from the other bins. Following the advice of Nair et al. [62], we use $N = 3$ bins for our effort datasets.

As a procedural detail, first we divided the data and then we applied the treatments. That is, all treatments saw the same training and test data.

3.3 Scoring Metrics

The results from each test set are evaluated in terms of two scoring metrics: magnitude of the relative error (MRE) [16] and Standardized Accuracy (SA). These scoring metrics are defined in Table 5. We use these since there are advocates for each in the literature. For example, Shepperd and MacDonell argue convincingly for the use of SA [75] (as well as for the use of effect size tests in effort estimation). Also in 2016, Sarro et al.[2] used MRE to argue their estimators were competitive with human estimates (which Molokken et al. [60] says lies within 30% and 40% of the true value).

Note that for these evaluation measures:

– MRE values: *smaller* are *better*

[2] http://tiny.cc/sarro16gecco

Table 6 Explanation of Scott-Knott test.

This study ranks methods using the Scott-Knott procedure recommended by Mittas & Angelis in their 2013 IEEE TSE paper [57]. This method sorts a list of l treatments with ls measurements by their median score. It then splits l into sub-lists m, n in order to maximize the expected value of differences in the observed performances before and after divisions. For example, we could sort $ls = 4$ methods based on their median score, then divide them into three sub-lists of of size $ms, ns \in \{(1,3),(2,2),(3,1)\}$. Scott-Knott would declare one of these divisions to be "best" as follows. For lists l, m, n of size ls, ms, ns where $l = m \cup n$, the "best" division maximizes $E(\Delta)$; i.e. the difference in the expected mean value before and after the spit:

$$E(\Delta) = \frac{ms}{ls} abs(m.\mu - l.\mu)^2 + \frac{ns}{ls} abs(n.\mu - l.\mu)^2$$

Scott-Knott then checks if that "best" division is actually useful. To implement that check, Scott-Knott would apply some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each half of the "best" division.
For a more specific example, consider the results from $l = 5$ treatments:

```
rx1 = [0.34, 0.49, 0.51, 0.6]
rx2 = [0.6,  0.7,  0.8,  0.9]
rx3 = [0.15, 0.25, 0.4,  0.35]
rx4=  [0.6,  0.7,  0.8,  0.9]
rx5=  [0.1,  0.2,  0.3,  0.4]
```

After sorting and division, Scott-Knott declares:

- Ranked #1 is rx5 with median= 0.25
- Ranked #1 is rx3 with median= 0.3
- Ranked #2 is rx1 with median= 0.5
- Ranked #3 is rx2 with median= 0.75
- Ranked #3 is rx4 with median= 0.75

Note that Scott-Knott found little difference between rx5 and rx3. Hence, they have the same rank, even though their medians differ.
Scott-Knott is prefered to, say, an all-pairs hypothesis test of all methods; e.g. six treatments can be compared $(6^2 - 6)/2 = 15$ ways. A 95% confidence test run for each comparison has a very low total confidence: $0.95^{15} = 46\%$. To avoid an all-pairs comparison, Scott-Knott only calls on hypothesis tests *after* it has found splits that maximize the performance differences.
For this study, our hypothesis test H was a conjunction of the A12 effect size test of and non-parametric bootstrap sampling; i.e. our Scott-Knott divided the data if *both* bootstrapping and an effect size test agreed that the division was statistically significant (95% confidence) and not a "small" effect ($A12 \geq 0.6$).
For a justification of the use of non-parametric bootstrapping, see Efron & Tibshirani [20, p220-223]. For a justification of the use of effect size tests see Shepperd & MacDonell [75]; Kampenes [33]; and Kocaguneli et al. [35]. These researchers warn that even if an hypothesis test declares two populations to be "significantly" different, then that result is misleading if the "effect size" is very small. Hence, to assess the performance differences we first must rule out small effects. Vargha and Delaney's non-parametric A12 effect size test explores two lists M and N of size m and n:

$$A12 = \left(\sum_{x \in M, y \in N} \begin{cases} 1 & \textit{if } x > y \\ 0.5 & \textit{if } x == y \end{cases} \right) / (mn)$$

This expression computes the probability that numbers in one sample are bigger than in another. This test was recently endorsed by Arcuri and Briand at ICSE'11 [5].

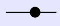
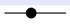
– SA values: *larger* are *better*

From the cross-vals, we report the *median* (termed *med*) which is the 50th percentile of the test scores seen in the *M*N results*. Also reported are the *inter-quartile range* (termed *IQR*) which is the (75-25)th percentile. The IQR is a non-parametric description of the variability about the median value.

For each datasets, the results from a *M*N-way* are sorted by their *median* value, then *ranked* using the Scott-Knott test recommended for ranking effort estimation experiments by Mittas et al. in TSE'13 [57]. For full details on Scott-Knott test, see Table 6. In summary, Scott-Knott is a top-down bi-clustering method that recursively divides sorted treatments. Division stops when there is only one treatment left or when a division of numerous treatments generates splits that are statistically *indistinguishable*. To judge when two sets of treatments are indistinguishable, we use a conjunction of *both* a 95% bootstrap significance test [20] *and* a A12 test for a non-small effect size difference in the distributions [55]. These tests were used since their non-parametric nature avoids issues with non-Gaussian distributions.

Table 7 shows an example of the report generated by our Scott-Knott procedure. Note that when multiple treatments tie for *Rank=1*, then we use the treatment's runtimes to break the tie. Specifically, for all treatments in *Rank=1*, we mark the faster ones as *Rank=1**.

Table 7 Example of Scott-Knott results. SA scores seen in the albrecht dataset. sorted by their median value. Here, *larger* values are *better*. **Med** is the 50th percentile and **IQR** is the *inter-quartile range*; i.e., 75th-25th percentile. Lines with a dot in the middle shows median values with the IQR. For the **Ranks**, *smaller* values are *better*. Ranks are computed via the Scott-Knot procedure from TSE13 [57]. Rows with the same ranks are statistically indistinguishable. 1* denotes rows of fastest best-ranked treatments.

		Standardized Accuracy		
Rank	**Method**	**Med**	**IQR**	
1	CART_FLASH	65	18	
1	CART_DE	59	19	
2	ABEN_DE	52	23	
2	ABE0	51	20	
2	RF	49	29	
2	LP4EE	47	25	
3	CART	41	31	
3	ABEN_RD	37	33	
3	ATLM	34	13	
3	CART_RD	32	27	
3	SVR	30	18	

3.4 Terminology for Optimizers

Some treatments are named "X_Y" which denote learner "X" tuned by optimizer "Y". In the following:

$$X \in \{CART, ABE\}$$
$$Y \in \{DE, RD, FLASH\}$$

Table 8 Average runtime (in minutes), for one-way out of an N*M cross-validation experiment. cross-validation (minutes). Executing on a 2GHz processor, with 8GB RAM, running Windows 10. Note that LP4EE and ATLM have no tuning results since the authors of these methods stress that it is advantageous to use their baseline methods, without any tuning. Last column reports totals for each dataset.

	faster									slower		
	ABE0	*CART*	*ATLM*	*LP4EE*	*SVR*	*RF*	*CART_RD*	*CART_DE*	*CART_FLASH*	*ABEN_RD*	*ABEN_DE*	total
kemerer	<1	<1	<1	<1	<1	<1	<1	<1	<1	4	5	13
albrecht	<1	<1	<1	<1	<1	<1	<1	<1	<1	4	6	15
finnish	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	6	18
miyazaki	<1	<1	<1	<1	<1	<1	<1	<1	<1	6	8	21
desharnais	<1	<1	<1	<1	<1	<1	<1	<1	<1	9	11	24
isbsg10	<1	<1	<1	<1	<1	<1	<1	<1	<1	7	10	23
maxwell	<1	<1	<1	<1	<1	<1	<1	<1	<1	12	16	34
kitchenham	<1	<1	<1	<1	<1	<1	<1	<1	<1	16	17	37
china	<1	<1	<1	<1	<1	<1	<1	<1	<1	23	26	54
total	3	4	4	4	5	7	8	9	8	86	105	

Note that we do not tune ATLM and LP4EE since they were designed to be used "off-the-shelf". Whigham et al. [87] declare that one of ATLM's most important features is that if does not need tuning. We also do not tune SVR and RF since we treat them as baseline algorithm-based methods in our benchmarks (i.e. use default settings in scikit-learn for these algorithms).

4 Results

4.1 Observations

Table 8 shows the runtimes (in minutes) for one of our N*M experiments for each dataset. From the last column of that table, we see that the median to maximum runtimes per dataset range are:

- 24 to 54 minutes, for one-way;
- Hence 8 to 18 hours, for the 20 repeats of our N*M experiments.

Performance scores for all datasets are shown in Table 9 and Table 10. We observe that ATLM and LP4EE performed as expected. Whigham et al. [87] and Sarro et al. [72] designed these methods to serve as baselines against which other treatments can be compared. Hence, it might be expected that in some cases these methods will perform comparatively below other methods. This was certainly the case here– as seen in Table 9 and Table 10, these baseline methods are top-ranked in only 8/18 datasets.

Another thing to observe in Table 9 and Table 10 is that random search (RD) also performed as expected; i.e. it was never top-ranked. This is a gratifying result since if random otherwise, then that tend to negate the value of hyperparameter optimization.

Table 9 *%* **MRE** from our cross-validation studies. *Smaller* values are *better*. Same format as Table 7. The gray rows show the *Rank=1+* results recommended for each data set. The phrase " *out-of-range* " denotes results that are so bad that they fall outside of the 0%..100% range shown here.

	Rank	**Method**	**Med.**	**IQR**	
albrecht	1	CART_FLASH	32	12	
	1	CART_DE	33	14	
	2	ABEN_DE	42	15	
	2	LP4EE	44	13	
	2	ABE0	45	16	
	2	RF	46	24	
	2	ABEN_RD	48	21	
	3	CART	53	22	
	3	CART_RD	54	20	
	3	SVR	56	19	
	4	ATLM	140	91	*out-of-range*
chine	1	LP4EE	45	5	
	1	ATLM	48	6	
	2	CART_FLASH	61	5	
	2	ABEN_DE	62	6	
	3	ABE0	64	5	
	3	CART_DE	64	6	
	4	ABEN_RD	68	7	
	4	RF	69	8	
	5	SVR	71	7	
	5	CART	71	5	
	5	CART_RD	72	6	
desharnais	1	CART_DE	35	11	
	1	CART_FLASH	35	11	
	1	LP4EE	38	13	
	2	RF	46	12	
	2	ABEN_DE	47	14	
	2	SVR	48	12	
	2	CART_RD	48	13	
	2	ABEN_RD	49	16	
	2	CART	49	14	
	2	ABE0	50	15	
	2	ATLM	54	17	
finnish	1	CART_FLASH	42	17	
	2	CART_DE	48	16	
	3	CART	57	21	
	3	RF	57	26	
	3	CART_RD	58	22	
	4	ABEN_DE	62	37	
	4	LP4EE	63	33	
	4	ABE0	64	48	
	4	ABEN_RD	64	42	
	4	SVR	74	13	
	5	ATLM	87	72	
isbsg10	1	CART_DE	59	20	
	1	CART_FLASH	62	19	
	2	SVR	72	17	
	2	CART_RD	73	24	
	2	ABE0	73	60	
	2	CART	74	21	
	2	ABEN_DE	74	25	
	2	LP4EE	75	23	
	2	ABEN_RD	76	35	
	2	RF	78	58	
	3	ATLM	127	124	*out-of-range*
kemerer	1	CART_DE	32	24	
	1	CART_FLASH	37	27	
	2	RF	50	39	
	2	ABEN_DE	54	22	
	2	LP4EE	54	23	
	2	CART_RD	55	25	
	2	CART	55	27	
	3	ABE0	56	33	
	3	SVR	59	14	
	3	ABEN_RD	60	17	
	4	ATLM	76	56	
kitchenham	1	CART_FLASH	34	6	
	1	ABEN_DE	35	8	
	2	LP4EE	38	6	
	2	CART_DE	38	7	
	2	ABE0	39	9	
	3	ABEN_RD	42	10	
	3	RF	43	8	
	4	CART	49	11	
	5	CART_RD	57	12	
	5	SVR	60	8	
	6	ATLM	106	108	*out-of-range*
maxwell	1	CART_FLASH	36	10	
	1	CART_DE	38	8	
	2	RF	51	15	
	2	LP4EE	51	16	
	2	CART	52	10	
	2	CART_RD	53	11	
	2	ABEN_DE	53	16	
	3	SVR	56	13	
	3	ABEN_RD	56	13	
	3	ABE0	56	14	
	4	ATLM	282	221	*out-of-range*
miyazaki	1	CART_DE	32	11	
	1	CART_FLASH	32	11	
	1	LP4EE	33	10	
	2	SVR	37	10	
	2	ATLM	37	32	
	2	ABEN_DE	39	16	
	3	RF	46	24	
	3	ABE0	47	12	
	3	CART	47	16	
	3	ABEN_RD	47	15	
	3	CART_RD	48	14	

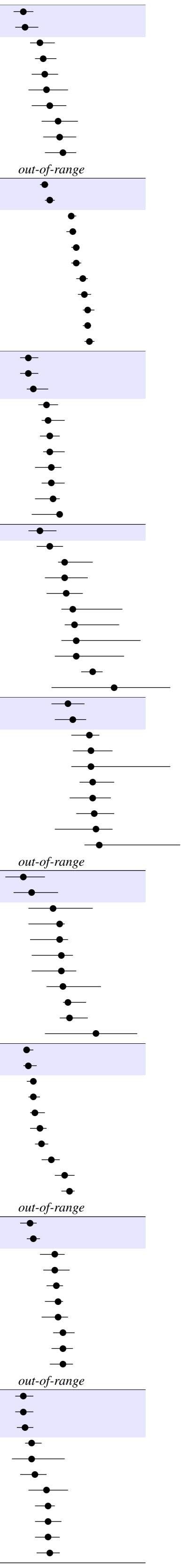
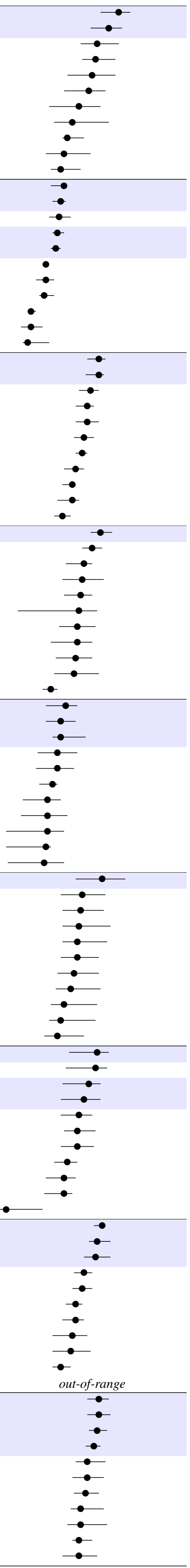


Table 10 *% SA* results from our cross-validation studies. *Larger* values are *better*. Same format as Table 7. The gray rows show the *Rank=1+* results recommended for each data set. The phrase " *out-of-range* " denotes results that are so bad that they fall outside of the 0%..100% range shown here.

	Rank	Method	Med.	IQR	
albrecht	1	CART_FLASH	65	18	
	1	CART_DE	59	19	
	2	ABEN_DE	52	23	
	2	ABE0	51	20	
	2	RF	49	29	
	2	LP4EE	47	25	
	3	CART	41	31	
	3	ABEN_RD	37	33	
	3	ATLM	34	13	
	3	CART_RD	32	27	
	3	SVR	30	18	
china	1	LP4EE	32	9	
	1	CART_FLASH	30	8	
	1	ABEN_DE	29	13	
	1	ABE0	28	7	
	1	CART_DE	27	6	
	2	SVR	21	3	
	2	RF	21	11	
	2	ABEN_RD	20	9	
	3	ATLM	12	5	
	3	CART	12	13	
	3	CART_RD	10	16	
desharnais	1	CART_FLASH	53	11	
	1	CART_DE	53	11	
	2	LP4EE	48	12	
	2	RF	46	11	
	2	ABEN_DE	46	14	
	2	ABE0	44	12	
	2	SVR	43	7	
	3	CART	39	12	
	3	ATLM	37	8	
	3	ABEN_RD	37	13	
	4	CART_RD	31	10	
finnish	1	CART_FLASH	54	13	
	2	CART_DE	49	12	
	3	RF	44	16	
	3	ABEN_DE	43	25	
	3	CART	42	17	
	4	ATLM	41	48	
	4	CART_RD	40	22	
	4	ABE0	40	25	
	4	LP4EE	39	22	
	4	ABEN_RD	38	27	
	5	SVR	24	9	
isbsg10	1	CART_DE	33	19	
	1	CART_FLASH	30	18	
	1	ATLM	30	20	
	2	ABEN_DE	28	24	
	2	ABE0	28	23	
	2	SVR	25	11	
	3	LP4EE	22	23	
	3	CART_RD	22	28	
	3	RF	22	35	
	3	ABEN_RD	21	27	
	3	CART	20	34	
kemerer	1	CART_DE	55	30	
	2	CART_FLASH	43	27	
	2	CART	42	25	
	2	RF	41	29	
	2	ABEN_DE	40	27	
	2	LP4EE	40	23	
	2	ABE0	38	25	
	2	CART_RD	36	27	
	3	ABEN_RD	32	28	
	3	ATLM	30	28	
	3	SVR	28	24	
kitchenham	1	LP4EE	52	24	
	1	ABEN_DE	51	25	
	1	ABE0	47	23	
	1	CART_FLASH	44	24	
	2	RF	41	19	
	2	ABEN_RD	40	19	
	2	CART_DE	40	20	
	3	CART	34	14	
	4	CART_RD	32	18	
	4	SVR	32	17	
	5	ATLM	-3	37	
maxwell	1	CART_FLASH	55	7	
	1	LP4EE	52	13	
	1	CART_DE	51	16	
	2	RF	44	11	
	2	ABEN_DE	43	12	
	3	ABE0	39	10	
	3	ABEN_RD	39	13	
	4	CART	37	21	
	4	CART_RD	36	23	
	4	SVR	30	11	
	5	ATLM	-107	99	*out-of-range*
miyazaki	1	CART_FLASH	53	13	
	1	CART_DE	53	14	
	1	LP4EE	52	11	
	1	ATLM	50	9	
	2	ABEN_DE	46	18	
	2	RF	46	19	
	2	ABE0	45	15	
	3	CART_RD	42	20	
	3	ABEN_RD	42	24	
	3	SVR	41	12	
	3	CART	41	21	



We also see in Table 9 empirical evidence many of our methods achieve human-competitive results. Molokken and Jorgensen [59]'s survey of current industry practices reports that human-expert predictions of project effort lie within 30% and 40% of the true value; i.e. MRE$\leq 40\%$. Applying that range to Table 9 we see that in 6/9 datasets, the best estimator has MRE$\leq 35\%$; i.e. they lie comfortably within the stated human-based industrial thresholds. Also, in a further 2/9 datasets, the best estimator has MRE$\leq 45\%$; i.e. they are close to the performance of humans.

The exception to the results in the last paragraph is isbg10 where the best estimator has an MRE$= 59\%$; i.e. our best performance is nowhere close to that of human estimators. In future work, we recommend researchers use isbg10 as a "stress test" on new methods.

4.2 Answers to Research Questions

Turning now to the research questions listed in the introduction:

RQ1: Is it best just to use the "off-the-shelf" defaults?

As mentioned in the introduction, Arcuri & Fraser note that for test case generation, using the default settings can work just as well as anything else. We can see some evidence of this effect in Table 9 and Table 10. Observe, for example, the kitchenham results where the untuned ABE0 treatment achieves *Rank=1**.

However, overall, Table 9 and Table 10 is negative on the use of default settings. For example, in datasets "albrecht", "desharnais", "finnish", not even one treatments that use the default found in *Rank=1**. Overall, if we always used just *one* of the methods using defaults (LP4EE, ATLM, ABE0) then that would achieve best ranks in 8/18 datasets.

Another aspect to note in the Table 9 and Table 10 results are the large differences in performance scores between the best and worst treatments (exceptions: miyazaki's MRE and SA scores do not vary much; and neither does isbg10's SA scores). That is, there is much to be gained by using the *Rank=1* treatments* and deprecating the rest.

In summary, using the defaults is recommended only in a part of datasets. Also, in terms of better test scores, there is much to be gained from tuning. Hence:

Lesson1: "Off-the-shelf" defaults should be deprecated.

RQ2: Can we replace the old defaults with new defaults?

If the hyperparameter tunings found by this paper were nearly always the same, then this study could conclude by recommending better values for default settings. This would be a most convenient result since, in future when new data arrives, the complexities of this study would not be needed.

Unfortunately, this turns out not to be the case. Table 11 shows the percent frequencies with which some tuning decision appears in our *M*N-way* cross validations (this table uses results from DE tuning CART since, as shown below, this usually leads to best results). Note that in those results it it not true that across most datasets there is a setting that is usually selected (thought min_samples_leaf less than 3 is often a popular setting). Accordingly, we say that Table 11 shows that there is much variations of the best tunings. Hence, for effort estimation:

Lesson2: Overall, there are no "best" default settings.

Before going on, one curious aspect of the Table 11 results are the %max_features results; it was rarely most useful to use all features. Except for finnish and china), best results were often obtained after discarding (at random) a quarter to three-quarters of the features. This is a clear indication that, in future work, it might be advantageous to explore more feature selection for CART models.

RQ3: Can we avoid slow hyperparameter optimization?

Some methods in our experiments (ABEN_RD and ABEN_DE) are slower than others, even with the same number of evaluations, as shown in Table 8. Is it possible to avoid such slow runtimes?

Long and slow optimization times are recommended when their exploration leads to better solutions. Such better solutions from slower optimizations are rarely found in Table 9 and Table 10 (only in 2/18 cases: see the ABE_DE results for kitchenha, and china). Further, the size of the improvements seen with the slower optimizers over the best Rank=2 treatments is small. Those improvements come at runtime cost (in Table 8), the kilo-optimizers are one orders of magnitude slower than other methods). Hence we say that for effort estimation:

Lesson3: Overall, our slowest optimizers perform no better than faster ones.

RQ4: What hyperparatmeter optimizers to use for effort estimation?

When we discuss this work with our industrial colleagues, they want to know "the bottom line"; i.e. what they should use or, at the very least, what they should

Table 11 Tunings discovered by hyperparameter selections (CART+DE). Table rows sorted by number of rows in data sets (smallest on top). Cells in this table show the percent of times a particular choice was made. White text on black denotes choices made in more than 50% of tunings.

	%max_features (selected at random; 100% means "use all")				max_depth (of trees)				min_sample_split (continuation criteria)				min_samples_leaf (termination criteria)			
	25%	50%	75%	100%	≤03	≤06	≤09	≤12	≤5	≤10	≤15	≤20	≤03	≤06	≤09	≤12
kemerer	18	32	23	27	57	37	05	00	95	02	03	00	92	02	05	02
albrecht	13	23	20	43	63	28	08	00	68	32	00	00	83	15	02	00
isbsg10	12	35	28	25	57	33	08	00	47	23	15	15	60	27	10	03
finnish	07	03	27	63	32	56	12	00	73	18	05	03	78	17	05	00
miyazaki	10	22	27	40	31	46	20	03	42	24	18	16	78	13	07	02
maxwell	04	16	40	40	18	60	20	02	44	27	17	12	50	33	14	04
desharnais	25	23	27	25	40	46	11	02	36	26	13	25	32	26	24	19
kitchenham	01	12	32	56	03	42	45	10	43	30	17	10	48	35	12	04
china	00	04	25	71	00	00	25	75	56	30	10	02	68	28	04	00

KEY: **10 20 30 40 50 60 70 80 90 100** %

not use. This section offers that advice. We stress that this section is based on the above results so, clearly these recommendations are something that would need to be revised whenever new results come to hand.

Table 12 lists the best that can be expected if an engineer chooses one of the estimators in our experiment, and applied it to all our datasets. The fractions shown at right come from counting optimizer frequencies in the top-ranks of Table 9 and Table 10. Note that the champion in our experiment is “CART_FLASH”, which ranked as ‘1’ in 16 out of all 18 cases. One close runner-up is “CART_DE”, which has 2 cases less in number of winning times. Those two estimators usually have good performance among most cases in the experiment.

Table 12 Methods ranking of total winning times (*Rank=1*), in all 18 experiment cases (9 datasets for both MRE and SA).

Rank	Method	Win Times
1	CART_FLASH	16/18
2	CART_DE	14/18
3	LP4EE	7/18
4	ATLM	3/18
5	ABEN_DE	2/18
5	ABE0	2/18
6	CART	0/18
6	ABEN_RD	0/18
6	RF	0/18
6	CART_RD	0/18
6	SVR	0/18

Beside the two top methods, none of the rest estimators could reach even half of all cases. Including those untuned baseline methods ($ATLM$, $LP4EE$). Hence, we cannot endorse their use for generating estimates to be shown to business managers. That said, we do still endorse their use as a baseline methods, for methodological reasons in effort estimation research (they are useful for generating a quick result against which we can compare other, better, methods).

Hence, based on the results of Table 12, for similar effort estimation tasks, we recommend:

Lesson4: For new datasets, try a combination of *CART* with the optimizers *differential evolution* and *FLASH*.

5 Threats to Validity

Internal Bias: Many of our methods contain stochastic random operators. To reduce the bias from random operators, we repeated our experiment in 20 times and applied statistical tests to remove spurious distinctions.

Parameter Bias: For other studies, this is a significant question since (as shown above) the settings to the control parameters of the learners can have a positive effect on the efficacy of the estimation. That said, recall that much of the technology of this paper concerned methods to explore the space of possible parameters. Hence we assert that this study suffers much less parameter bias than other studies.

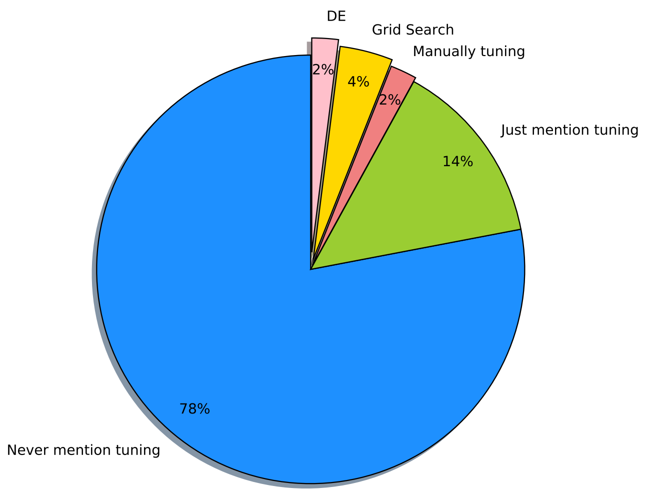
Sampling Bias: While we tested OIL on the nine datasets, it would be inappropriate to conclude that OIL tuning always perform better than others methods for all datasets. As researchers, what we can do to mitigate this problem is to carefully document out method, release out code, and encourage the community to try this method on more datasets, as the occasion arises.

6 Related Work

In software engineering, hyperparameter optimization techniques have been applied to some sub-domains, but yet to be adopted in many others. One way to characterize this paper is an attempt to adapt recent work in hyperparameter optimization in software defect prediction to effort estimation. Note that, like in defect prediction, this article has also concluded that Differential Evolution is an useful method.

Several SE defect prediction techniques rely on static code attributes [44, 64, 82]. Much of that work has focused of finding and employing complex and "off-the-shelf" machine learning models [54, 61, 21], without any hyperparameter optimization. According to a literature review done by Fu et al. [25], as shown in Figure 2, nearly 80% of highly cited papers in defect prediction do not mention parameters tuning (so they rely on the default parameters setting of the predicting models).

Fig. 2 Literature review of hyperparameters tuning on 52 top defect prediction papers [25]



Gao et al. [26] acknowledged the impacts of the parameter tuning for software quality prediction. For example, in their study, "*distanceWeighting*" parameter was set to "*Weight by 1/distance*", the *KNN* parameter "*k*" was set to "30", and the "*cross-Validate*" parameter was set to "*true*". However, they did not provide any further explanation about their tuning strategies.

As to methods of tuning, Bergstra and Bengio [9] comment that *grid search*[3] is very popular since (a) such a simple search to gives researchers some degree of insight; (b) grid search has very little technical overhead for its implementation; (c) it is simple to automate and parallize; (d) on a computing cluster, it can find better tunings than sequential optimization (in the same amount of time). That said, Bergstra and Bengio deprecate grid search since that style of search is not more effective than more randomized searchers if the underlying search space is inherently low dimensional.

[3] For N tunable option, run N nested for-loops to explore their ranges.

This remark is particularly relevant to effort estimation since datasets in this domain are often low dimension [41].

Lessmann et al. [47] used grid search to tune parameters as part of their extensive analysis of different algorithms for defect prediction. However, they only tuned a small set of their learners while they used the default settings for the rest. Our conjecture is that the overall cost of their tuning was too expensive so they chose only to tune the most critical part.

Two recent studies about investigating the effects of parameter tuning on defect prediction were conducted by Tantithamthavorn et al. [83, 84] and Fu et al. [24]. Tantithamthavorn et al. also used grid search while Fu et al. used differential evolution. Both of the papers concluded that tuning rarely makes performance worse across a range of performance measures (precision, recall, etc.). Fu et al. [24] also report that different datasets require different hyperparameters to maximize performance.

One major difference between the studies of Fu et al. [24] and Tantithamthavorn et al. [83] was the computational costs of their experiments. Since Fu et al.'s differential evolution based method had a strict stopping criterion, it was significantly faster.

Note that there are several other methods for hyperparameter optimization and we aim to explore several other method as a part of future work. But as shown here, it requires much work to create and extract conclusions from a hyperparameter optimizer. One goal of this work, which we think we have achieved, to identify a simple baseline method against which subsequent work can be benchmarked.

7 Conclusions and Future Work

Hyperparameter optimization is known to improve the performance of many software analytics tasks such as software defect prediction or text classification [1, 2, 24, 84]. Most prior work in this effort estimation optimization only explored very small datasets [48] or used estimators that are not representative of the state-of-the-art [87, 72]. Other researchers assume that the effort model is a specific parametric form (e.g. the COCOMO equation), which greatly limits the amount of data that can be studied. Further, all that prior work needs to be revisited given the existence of recent and very prominent methods; i.e. ATLM from TOSEM'15 [87] and LP4EE from TOSEM'18 [72].

Accordingly, this paper conducts a more thorough investigation of hyperparameter optimization for effort estimation using methods (a) with no data feature assumptions (i.e. no COCOMO data); (b) that vary many parameters (6,000+ combinations); that tests its results on 9 different sources with data on 945 software projects; (c) which uses optimizers representative of the state-of-the-art (DE [81], FLASH [63]); and which (d) benchmark results against prominent methods such as ATLM and LP4EE.

These results were assessed with respect to the Arcuri and Fraser's concerns mentioned in the introduction; i.e. sometimes hyperparamter optimization can be both too slow and not effective. Such pessimism may indeed apply to the test data generation domain. However, the results of this paper show that there exists other domains like effort estimation where hyperparameter optimization is both useful and fast. After applying hyperparamter optimization, large improvements in effort estimation accuracy were observed (measured in terms of the standardized accuracy). From those results, we can recommend using a combination of regression trees (CART) tuned by different evolution and FLASH. This particular combination of learner and optimizers can achieve in a few minutes what other optimizers need longer runtime of CPU to accomplish.

To the best of our knowledge, this study is one of the most extensive exploration of hyperparameter optimization and effort estimation yet undertaken. That said, there are very many options not explored here. Our current plans for future work include the following.

– Try other learners: e.g. neural nets, bayesian learners or AdaBoost;
– Try other data pre-processors. We mentioned above how it was curious that max features was often less than 100%. This is a clear indication that, we might be able to further improve our estimations results by adding more intelligent feature selection to, say, CART.
– Other optimizers. For example, combining DE and FLASH might be a fruitful way to proceed.
– Yet another possible future direction could be hyper-hyperparamter optimization. In the above, we used optimizers like differential evolution to tune learners. But these optimizers have their own control parameters. Perhaps there are better settings for the optimizers? Which could be found via hyper-hyperparameter optimization?

Hyper-hyperparameter optimization could be a very slow process. Hence, results like this paper could be most useful since here we have identified optimizers that are very fast and very slow (and the latter would not be suitable for hyper-hyperparamter optimization).

In any case, we hope that OIL and the results of this paper will prompt and enable much more research on better methods to tune software effort estimators. To that end, we have placed all our scripts and data on-line at https://github.com/ai-se/magic101

Acknowledgement

This work was partially funded by a National Science Foundation Award 1703487.

References

1. Agrawal A, Menzies T (2018) " better data" is better than" better data miners"(benefits of tuning smote for defect prediction). In: ICSE'18
2. Agrawal A, Fu W, Menzies T (2018) What is wrong with topic modeling? and how to fix it using search-based software engineering. IST Journal
3. Aljahdali S, Sheta AF (2010) Software effort estimation by tuning coocmo model parameters using differential evolution. In: Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on, IEEE, pp 1–6

4. Angelis L, Stamelos I (2000) A simulation tool for efficient analogy based cost estimation. EMSE 5(1):35–68
5. Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Software Engineering (ICSE), 2011 33rd International Conference on, IEEE, pp 1–10
6. Arcuri A, Fraser G (2013) Parameter tuning or default values? an empirical investigation in search-based software engineering. ESE 18(3):594–623
7. Atkinson-Abutridy J, Mellish C, Aitken S (2003) A semantically guided and domain-independent evolutionary model for knowledge discovery from texts. IEEE Transactions on Evolutionary Computation 7(6):546–560
8. Baker DR (2007) A hybrid approach to expert and model based effort estimation. West Virginia University
9. Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(1):281–305
10. Boehm BW (1981) Software engineering economics. Prentice-Hall
11. Breiman L (2001) Random forests. Machine learning 45(1):5–32
12. Briand LC, Wieczorek I (2002) Resource estimation in software engineering. Encyclopedia of software engineering
13. Chalotra S, Sehra SK, Brar YS, Kaur N (2015) Tuning of cocomo model parameters by using bee colony optimization. Indian Journal of Science and Technology 8(14)
14. Chang CC, Lin CJ (2011) Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST) 2(3):27
15. Chang CL (1974) Finding prototypes for nearest neighbor classifiers. TC 100(11)
16. Conte SD, Dunsmore HE, Shen VY (1986) Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA
17. Corazza A, Di Martino S, Ferrucci F, Gravino C, Sarro F, Mendes E (2013) Using tabu search to configure support vector regression for effort estimation. Empirical Software Engineering 18(3):506–546
18. Cowing K (2002) Nasa to shut down checkout & launch control system. http://www.spaceref.com/news/viewnews.html?id=475
19. Das S, Wong W, Dietterich T, Fern A, Emmott A (2016) Incorporating expert feedback into active anomaly discovery. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp 853–858, DOI 10.1109/ICDM.2016.0102
20. Efron B, Tibshirani J (1993) Introduction to bootstrap. Chapman & Hall
21. Elish KO, Elish MO (2008) Predicting defect-prone software modules using support vector machines. J Syst Softw 81(5):649–660, DOI 10.1016/j.jss.2007.07.040
22. Foss T, Stensrud E, Kitchenham B, Myrtveit I (2003) A simulation study of the model evaluation criterion mmre. TSE 29(11):985–995
23. Frank E, Hall M, Pfahringer B (2002) Locally weighted naive bayes. In: 19th conference on Uncertainty in Artificial Intelligence, pp 249–256
24. Fu W, Menzies T, Shen X (2016) Tuning for software analytics: Is it really necessary? IST Journal 76:135–146
25. Fu W, Nair V, Menzies T (2016) Why is differential evolution better than grid search for tuning defect predictors? arXiv preprint arXiv:160902613

26. Gao K, Khoshgoftaar TM, Wang H, Seliya N (2011) Choosing software metrics for defect prediction: an investigation on feature selection techniques. Software: Practice and Experience 41(5):579–606
27. Hall MA, Holmes G (2003) Benchmarking attribute selection techniques. TKDE 15(6):1437–1447
28. Hardy MA (1993) Regression with dummy variables, vol 93. Sage
29. Hihn J, Menzies T (2015) Data mining methods and cost estimation models: Why is it so hard to infuse new ideas? In: ASEW, pp 5–9, DOI 10.1109/ASEW.2015.27
30. Jørgensen M (2004) A review of studies on expert estimation of software development effort. JSS 70(1-2):37–60
31. Jørgensen M (2015) The world is skewed: Ignorance, use, misuse, misunderstandings, and how to improve uncertainty analyses in software development projects
32. Jørgensen M, Gruschke TM (2009) The impact of lessons-learned sessions on effort estimation and uncertainty assessments. TSE 35(3):368–383
33. Kampenes VB, Dybå T, Hannay JE, Sjøberg DI (2007) A systematic review of effect size in software engineering experiments. Information and Software Technology 49(11-12):1073–1086
34. Kemerer CF (1987) An empirical validation of software cost estimation models. CACM 30(5):416–429
35. Keung J, Kocaguneli E, Menzies T (2013) Finding conclusion stability for selecting the best effort predictor in software effort estimation. ASE 20(4):543–567, DOI 10.1007/s10515-012-0108-5
36. Keung JW, Kitchenham BA, Jeffery DR (2008) Analogy-x: Providing statistical inference to analogy-based software cost estimation. TSE 34(4):471–484
37. Kitchenham BA, Pickard LM, MacDonell SG, Shepperd MJ (2001) What accuracy statistics really measure. IEEE Software 148(3):81–85
38. Kocaguneli E, Menzies T (2011) How to find relevant data for effort estimation? In: ESEM, pp 255–264, DOI 10.1109/ESEM.2011.34
39. Kocaguneli E, Misirli AT, Caglayan B, Bener A (2011) Experiences on developer participation and effort estimation. In: SEAA'11, IEEE, pp 419–422
40. Kocaguneli E, Menzies T, Bener A, Keung JW (2012) Exploiting the essential assumptions of analogy-based effort estimation. TSE 38(2):425–438
41. Kocaguneli E, Menzies T, Keung J, Cok D, Madachy R (2013) Active learning and effort estimation: Finding the essential content of software effort estimation data. IEEE Transactions on Software Engineering 39(8):1040–1053
42. Kocaguneli E, Menzies T, Mendes E (2015) Transfer learning in effort estimation. ESE 20(3):813–843, DOI 10.1007/s10664-014-9300-5
43. Korte M, Port D (2008) Confidence in software cost estimation results based on mmre and pred. In: PROMISE'08, pp 63–70
44. Krishna R, Menzies T, Fu W (2016) Too much automation? the bellwether effect and its implications for transfer learning. In: IEEE/ACM ICSE, ASE 2016, DOI 10.1145/2970276.2970339
45. Langdon WB, Dolado J, Sarro F, Harman M (2016) Exact mean absolute error of baseline predictor, marp0. IST 73:16–18

46. LBreiman ROCS J Friedman (1984) Classification and Regression Trees. Wadsworth
47. Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 34(4):485–496, DOI 10.1109/TSE.2008.35
48. Li Y, Xie M, Goh TN (2009) A study of project selection and feature weighting for analogy based software cost estimation. JSS 82(2):241–252
49. McConnell S (2006) Software estimation: demystifying the black art. Microsoft press
50. Mendes E, Mosley N (2002) Further investigation into the use of cbr and stepwise regression to predict development effort for web hypermedia applications. In: ESEM'02, IEEE, pp 79–90
51. Mendes E, Watson I, Triggs C, Mosley S N Counsell (2003) A comparative study of cost estimation models for web hypermedia applications. ESE 8(2):163–196
52. Menzies T, Zimmermann T (2018) Software analytics: Whats next? IEEE Software 35(5):64–70
53. Menzies T, Chen Z, Hihn J, Lum K (2006) Selecting best practices for effort estimation. TSE 32(11):883–895
54. Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33(1):2–13, DOI 10.1109/TSE.2007.256941
55. Menzies T, Yang Y, Mathew G, Boehm B, Hihn J (2017) Negative results for software effort estimation. ESE 22(5):2658–2683, DOI 10.1007/s10664-016-9472-2
56. Menzies T, Majumder S, Balaji N, Brey K, Fu W (2018) 500+ times faster than deep learning:(a case study exploring faster methods for text mining stackoverflow). In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), IEEE, pp 554–563
57. Mittas N, Angelis L (2013) Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Trans SE 39(4):537–551, DOI 10.1109/TSE.2012.45
58. Moeyersoms J, Junqué de Fortuny E, Dejaeger K, Baesens B, Martens D (2015) Comprehensible software fault and effort prediction. J Syst Softw 100(C):80–90, DOI 10.1016/j.jss.2014.10.032
59. Molokken K, Jorgensen M (2003) A review of software surveys on software effort estimation. In: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., pp 223–230, DOI 10.1109/ISESE.2003.1237981
60. Molokken K, Jorgensen M (2003) A review of software surveys on software effort estimation. In: Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on, IEEE, pp 223–230
61. Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: 30th ICSE, DOI 10.1145/1368088.1368114
62. Nair V, Agrawal A, Chen J, Fu W, Mathew G, Menzies T, Minku LL, Wagner M, Yu Z (2018) Data-driven search-based software engineering. In: MSR

63. Nair V, Yu Z, Menzies T, Siegmund N, Apel S (2018) Finding faster configurations using flash. IEEE Transactions on Software Engineering pp 1–1, DOI 10.1109/TSE.2018.2870895
64. Nam J, Kim S (2015) Heterogeneous defect prediction. In: 10th FSE, ESEC/FSE 2015, DOI 10.1145/2786805.2786814
65. Neter J, Kutner MH, Nachtsheim CJ, Wasserman W (1996) Applied linear statistical models, vol 4. Irwin Chicago
66. Olsson F (2009) A literature survey of active machine learning in the context of natural language processing. Tech. Rep. T2009:06, URL `http://soda.swedish-ict.se/3600/1/SICS-T--2009-06--SE.pdf`
67. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J (2011) Scikit-learn: Machine learning in python. JMLR 12(Oct):2825–2830
68. Peters T, Menzies T, Layman L (2015) Lace2: Better privacy-preserving data sharing for cross project defect prediction. In: ICSE, vol 1, pp 801–811, DOI 10.1109/ICSE.2015.92
69. Port D, Korte M (2008) Comparative studies of the model evaluation criterion mmre and pred in software cost estimation research. In: ESEM'08, pp 51–60
70. Quinlan JR (1992) Learning with continuous classes. In: 5th Australian joint conference on artificial intelligence, Singapore, vol 92, pp 343–348
71. Rao GS, Krishna CVP, Rao KR (2014) Multi objective particle swarm optimization for software cost estimation. In: ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I, Springer, pp 125–132
72. Sarro F, Petrozziello A (2018) Linear programming as a baseline for software effort estimation. ACM Transactions on Software Engineering and Methodology (TOSEM) p to appear
73. Sarro F, Petrozziello A, Harman M (2016) Multi-objective software effort estimation. In: ICSE, ACM, pp 619–630
74. Shepperd M (2007) Software project economics: a roadmap. In: 2007 Future of Software Engineering, IEEE Computer Society, pp 304–315
75. Shepperd M, MacDonell S (2012) Evaluating prediction systems in software project estimation. IST 54(8):820–827
76. Shepperd M, Schofield C (1997) Estimating software project effort using analogies. TSE 23(11):736–743
77. Shepperd M, Cartwright M, Kadoda G (2000) On building prediction systems for software engineers. EMSE 5(3):175–182
78. Singh BK, Misra A (2012) Software effort estimation by genetic algorithm tuned parameters of modified constructive cost model for nasa software projects. International Journal of Computer Applications 59(9)
79. Sommerville I (2010) Software engineering. Addison-Wesley
80. Stensrud E, Foss T, Kitchenham B, Myrtveit I (2003) A further empirical investigation of the relationship of mre and project size. ESE 8(2):139–161
81. Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over cont. spaces. JoGO 11(4):341–359

82. Tan M, Tan L, Dara S (2015) Online defect prediction for imbalanced data. In: ICSE
83. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: 38th ICSE, DOI 10.1145/2884781.2884857
84. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The impact of automated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering pp 1–1, DOI 10.1109/TSE.2018.2794977
85. Trendowicz A, Jeffery R (2014) Software project effort estimation. Foundations and Best Practice Guidelines for Success, Constructive Cost Model–COCOMO pags pp 277–293
86. Walkerden F, Jeffery R (1999) An empirical study of analogy-based software effort estimation. ESE 4(2):135–158
87. Whigham PA, Owen CA, Macdonell SG (2015) A baseline model for software effort estimation. TOSEM 24(3):20:1–20:11, DOI 10.1145/2738037

Replies to Associate Editor Comments

Thank you for this opportunity to revise the previous draft. Based on reviewer feedback, we have made many improvements:

- Based on reviewer comments we have made many, small (but important) clarifications, corrections, and (sometimes) deletions. For full details on those changes, see below.
- Better choice of optimizers:
  - Reviewers noted that (a) it is somewhat strange to use multi-objective optimizers for the single objective problem studied here; (b) we did not apply our own state-of-the-art method (FLASH) to this task.
  - Hence, in this draft, we use FLASH and have removed the multi-obective optimiozers.
- Reviewers notes that we missed some prominent learners (Random Forest and Support Vector Regression). They have now been added to our analysis.
- Better use of data. At reviewer2's suggestion, we deleted certain inappropriate attributes;

We hope this draft finds favor with the reviewers,

Replies to Reviewer 1 Comments

You made several points about poor choice of wordings. We agree with all those points and have deleted the required text.

The rest of this section addresses your longer substantive comments.

I appreciate that these comments are intended as blind but I feel it is in all our interests to identify myself so that my comments - particularly regarding ATLM - can be fully understood. So in the spirit of full disclosure, I am Stephen MacDonell, initiator and co-author of the ATLM study, and also second author of the Standardised Accuracy paper.

Pleased to be working with you.

Summary comments:

This paper seeks to systematically evaluate the potential effect parameter tuning could have on the relative accuracy of project-level effort estimation models, also taking into account the time required to perform that tuning. A small but reasonable number of modelling and parameter optimization methods are compared, including two recently proposed baseline approaches, using a test rig created for that purpose and (very positively) made available by the authors. The empirical analysis appears to have been competently executed, with reasoned justification at each decision point. While the results are somewhat mixed, the authors conclude that tuning is important in this context and that a particular combination of modelling and parameter optimization methods should be tried by those seeking to estimate effort based on existing data sets.

Please note that based on remarks from reviewer2, we have a slightly different rig. And now our results are less mixed, more definitive.

While on balance - and in spite of my many comments - I am positive about this paper, I do have to wonder whether the underlying premise of this research - that effort estimation is conducted up front at the project level - is in fact a contemporary concern.

You list below many of those concerns and, we think, we can offer replies on those matters.

But, respectfully, we think that you want this to be another paper. Where we 100% agree with you is that there is always value in a basic re-evaluation of the premises to a field. And, in the case of effort estimation, that re-evaluation could go along the lines that you suggest. But in the terminology of Kuhn, that would be a paradigm smasher rather than a result built using the assumptions of a current paradigm.

Sp you seem to want a paradigm smasher and this paper is not a paradigm smasher. Rather, it is the next step in a research program that dates back to Boehm (1981), Shepperd (1997) et al. We think that it is a very large step (particularly with respect to the new FLASH algorithm introduced in this draft) so our belief is that that step merits journal-level publication.

Now we rush to add that we mean no disrespect to your opinions. And we hope you see in the following text that we take your concerns seriously.

In an era characterised by noprojects and noestimates what is the continued relevance of project-level effort estimation based on historical data? And who has the volume of data that would enable such analyses? The requirement by government for such an approach, which I'm not challenging, says more about their dated practices than it does about contemporary thinking re forecasting and resource allocation.

We take it this comment is not a fatal flaw in this current paper since, in many ways, the motives and methods of this paper are very similar to those of your own work (as witnessed by your ATLM publication at TOSEM).

That said, to reply to the specifics of your comment, we've read much about noestimation and I think if 100% of my staff were Silicon Valley developers motivated by $300K/year packages and huge stock shares in the company, and all those people say themselves as CEOs-in-training, then I'd be happy to through out schedules and declare "ok everyone, to hell with effort models. Just add value where you can, as fast as you can".

But we've meet those people, 5 years later, after they've relocated to Raleigh (since their expansion plans were thwarted since they could not hire or retain the people they want). Now, in their new life, they themselves may still be a mountain of talent but they are surrounded by a very large landscape of developers of widely different motivations, and skills. And in that new life, as they struggle to bring their development team under some degree of control and predictability, suddenly effort estimation is look more and more interesting to them.

Detailed comments:

0. Abstract -

R1_01: - Software analytics are highly prominent in software engineering _research_ but whether they are "widely used" in practice remains an open question (particularly in relation to the point made next).

Quite correct... that claim needs a reference (see first sentence of Introduction of new draft). For the recent IEEE Software special issue on 50 years of SE, Menzies and Zimmermann wrote a short paper defending exactly that claim [52].

R1_02: - (As noted above) Given the predominance of agile and hybrid development methods, what contexts continue to require up-front project-level estimation of effort? And to what extent is this a minor or major component of the SE industry's work?

We quite take your point that estimation-as-conceived- by-Bohem was a very waterfall, approach. And we need to adapt more to changing approaches. We have much research in that area, eg. sse Menzies last TSE paper on deep learning and story points. While that result is certainly useful, the computational cost of DL is a open research area that we must resolve, This paper is a stepping stone to that work. Here we show that many methods in widespread use can be replaced by much simple,must faster methods. Of course, our next paper should take the step but before moving on, let consolidate what we have.

Also, we note that in some domain traditional domain (like bank or tax system), agile and hybrid development methods cannot be applied, but we still need to do the up-front estimation for them. There are many SE industry still need traditional but safer ways to development, so its a major component.

R1_03: - "Accordingly, this paper seeks simple, automatic, effective and fast methods for finding good tunings for automatic software effort estimation." This is all fine as comprising the Abstract but each of these expectations is both important and vague - each one needs to be operationalised (simple, automatic, effective, fast, good), and most are somewhere in the paper. Personally I'd probably prefer to see some sense of their meanings captured/represented here but can understand the tension this places on an Abstract.

Good point- as you say, we wont add this to the abstract but will add these definitions to the abstract (and revisit these in the conclusion). Simple: easy to code Automatic: no need to manually set the hyperparameters Effective: high performance in experiment Fast: Running time is fast Good: a comprehensive summary

R1_04: - "This particular combination of learner and optimizers often achieves in a few hours what other optimizers need days to weeks of CPU time to accomplish." With all due respect to Arcuri (note: not "Acuri") and Fraser, is this our biggest concern? (As noted above) How accurate is accurate enough for practice? How fast is fast enough for practice? "It is hoped that this paper will prompt and enable much more research on better methods to tune software effort estimators." Is this what practice is demanding? Fundamentally software engineering is a practice-based discipline

Typo in Arcuris name? That is very bad! Thanks for pointing that out

The accuracy we discussed here is to get rank as 1*, we focus on to get best performance among the methods we have. "Enough" is a ratio word where we comapre our performance to others.

And as for what industry is demanding effort estimation, we appeal to Jorgensen who reported to the CREST open workshop in 20115 that:

– The world is skewed: Ignorance, use, misuse, misunderstandings, and how to improve uncertainty analyses in software development projects
http://crest.cs.ucl.ac.uk/cow/44/slides/cow44_Jorgensen.pdf

Jorgensen notes that what mangers need is not point estimates. Rather they need ranges derived via "what-if queries" that cover the range of uncertainty within a project. He noted that such what if queries are hard to do with humans (too many what-ifs) but it can be done with models (perturb their input space, 1000s of times). We have tools to to that (see http://menzies.us/pdf/xomo-sa-v6.pdf). This kinds of analysis only works (and can be trusted) if there is a certified model in the middle of the Monte Carlo simulation. So this is entire paper can be viewed as a validation procedure for one part of a Monte Carlo simulator.

0. Keywords -

R1_05: - I do not feel that "Evolutionary Algorithms" should be a keyword here - they are used peripherally; why not say "Analogy"? "Regression Trees"? This just needs a bit more thought.

Good point. Keywords revised.

1. Introduction -

R1_06: - We do not (always) motivate our research particularly well. For instance: "This is needed since software effort estimates can be wildly inaccurate [32]." Like citing the widely criticized Standish reports, this is a rather weak statement, as it can be interpreted equally that estimation can also be highly accurate, and reference [32] is regrettably from 1987. The works cited in support of the criticality of effort estimation are also now rather dated, reflecting past interest in project-level estimation, and including the generic texts by McConnell and Sommerville. Furthermore a "poor estimate" in terms of accuracy wrt the actual can arise for numerous, non-estimation, reasons - cultural challenges, political influence, power structures and legislative changes among them... Yet we tend to gloss over these and just say "Bad Estimates".

Good point, we revise and delete the Standish reference. Please see our new para1 in the introduction.

R1_07: - Prior comments regarding "how fast" and "how accurate" notwithstanding, I actually like the four research questions that are addressed here. I was a little confused regarding the reference in RQ3 to "whenever new data arrives" though - is this new data for an existing data set, e.g., a new project being added on chronologically, or an entirely new data set being brought to the modelling process? (Or something else again that I'm missing?)

When a new project is initialized, we will have new related data for estimation, so more likely the new data will be different from original dataset.

R1_8: - Here and later in the paper the conclusion is extrapolated: "there exists (b) other domains (e.g., effort estimation)" - as domains other than effort estimation have not been investigated here then better to just state that, rather than speculate that the outcome might be more broadly applicable.

You are correct. We have revised. Please see at the end of Section1.

2. About Effort Estimation -

R1_9 - ”most realistic amount of human effort” - one, what does ”most realistic” mean? two, and as noted many years ago by Glass, estimation is not necessarily a rational process.

You are right. We now changed from Realistic to approximate, see para1 of section2

R1_10: - Do we have evidence that good estimation means waste is avoided? Again I feel it is unfortunate that the example used in this section introduction is from 2002.

Good point- we have added more references here.

R1_11: - The discussion of story points is incorrect. Story points are intentionally not a measure of the ”overall effort that will be required to fully implement a piece of work” - they are team-specific indicators of the relative size of stories or tasks. The paragraph beginning ”It should be noted that planning poker...” is presented without any credible evidence in support of the statements made. I have to say that when the section went on to basically states that human-based estimation methods were then being disregarded in the study I wondered why they should have any coverage at all. It may be better to simply state that the paper is only concerned with algorithmic methods, due to the need for auditability, and leave it at that. (This would mean you could leave out subsection 2.1 which, for me, is currently problematic.)

Good point- we have deleted that section

R1_12: - The other aim we had in providing ATLM, apart from it being simple and so requiring no decisions re parameters from the user, was that it could be applied automatically - and as such there should be no excuse not to compare any new model against a comparatively naive baseline.

Good point- we have added the following text to the description of ATLM.

One aim of ATLM is that apart from it being simple and so requiring no decisions re parameters from the user, is that it can be applied automatically - and as such there should be no excuse not to compare any new model against a comparatively naive baseline.

R1_13: - ”Typically, this removes a small percentage of the training data.” Any evidence for this?

Good point- deleted.

R1_14: - ”...then realized that of the earlier advocates of TS...” I suspect there is a missing ”none” in this clause.

Good point-added

R1_15: - In the caption for Figure 1 ”optimal” should be ”optional”.

Fixed

R1_16: - I commend the development and release of OIL and supporting materials.

Thank you for those kind words.

3. Empirical Study -

R1_17: - Overall I found the description of the empirical study easy to follow and while choices were inevitably made along the way these were justified sufficiently using prior evidence.

Thank you for those kind words.

R1_18: - ”We represent results in terms of SA since this measure has been adopted in recent high-profile publications [41, 67].” If you want to retain this statement it needs to be strengthened - [41] was the paper that proposed SA and [67] was a refinement of the approach; what is needed is studies that have used it in anger.

This reviewer will be happy to note that in the last five years of publication, when we have used SA in our publications, no reviewer has ever questioned it (though they have questioned older measure such MMRE). We recommend this reviewer smiles and be happy that their work is so influential in the estimation arena.

4. Results - R1_19: - It is probably worth noting in regard to the results shown in Table 8 that the slower performance of the models using ABEN and NSGAII was not unexpected given we know how they work and that they are inherently slower than the other approaches/tunings.

Good point: based on comments from reviewer2, we re-ran all our experiments and removed unfair comparison with NSGA-II.

R1_20: - The fact that ATLM and/or LP4EE actually achieved top-ranked results for 3 of the 9 datasets is telling given these are very low-threshold baselines that require minimal effort and computation - I feel something ought to be said about this in terms of the results.

Sadly, in the new experiments and using one more optimizer (FLASH), we can no longer see ATLM working so well (see Table12).

R1_25: - I very much like the depiction in Table 10.

Thank you for those kind words.

R1_21: - ”Further, the size of the improvements seen with kilo-optimizers over the best Rank=2 treatments is very small. Those improvements come at significant runtime cost...” I would be cautious about using terms such as ”very small” and ”significant” here. (Also see ”dramatically” in Section 7).

Good point: all drama removed. Also ”Very” and ”significant” deleted.

6. Related Work - R1_22: - Re the low dimensionality of search spaces it would be worth pointing out that this is the case with respect to effort estimation data sets - especially if we do the pre-processing as noted above.

Good point: text changed as you suggested. Reference to low dimensionality of effort data added.

7. Conclusions and Future Work - R1_29: - For me the results were not as definitive as their depiction here in terms of the superiority of the modelling/tuning: ”After applying hyperparameter optimization, large improvements in effort estimation accuracy were observed (measured in terms of the standardized accuracy).” Given the relatively good performance for ATLM and LP4EE in 3 of 9 cases I feel that the case for complex modelling and tuning is not as strong as is indicated here.

With the new experiments, we think we have a much better case now.

R1_30: - Regarding the proposed directions for future work I’d like to see some rationale for the proposals - at present it comes across as opportunistic, a sort of ’because we can, we should’ approach. What is the basis in theory or in prior empirical evidence for the possibilities listed?

Good point- we not take more care to justify our future work in terms of the current threats to validity.

Replies to Reviewer2 Comments

Thank you for the detail of your review. It was most insightful and helpful.

We offer detailed notes on your comments below. Before reviewing note, please note the major changes in this draft (all of which were prompted by your comments):

- You noted that (a) it is somewhat strange to use multi-objective optimizers for the single objective problem studied here; (b) we did not apply our own state-of-the-art method (FLASH) to this task.
- Hence, in this draft, we use FLASH and have removed the multi-obective optimiozers.

You noted that we missed some prominent learners (Random Forest and Support Vector Regression). They have now been added to our analysis.

Better use of data. At your suggestion, we deleted certain inappropriate attributes;

The paper investigates an approach for machine learning hyperparameters optimization, named OIL (Optimized Inductive Learning), and empirically evaluate its performance in the context of software development effort estimation. To this end 9 public datasets, 2 machine learners and 7 optimizers are used. The results confirm previous findings that tuning is necessary to obtain improved estimations and that some combinations of learner + optimizers are better than others for the datasets analysed.

The aim of the work is interesting and the topic worth of investigation; however, the current manuscript is not yet mature for publication due to serious weaknesses which need to be carefully addressed, as detailed below.

R2_01: First and foremost, the empirical study seems flawed. Indeed, many of the variables used as cost drivers to build the prediction model cannot be used for effort estimation purposes as they are highly correlated or overlap (e.g. Adjusted function points and raw function point) or computed based on the effort values such as in the case of the variables duration and productivity, or not available at prediction time (e.g. KSLOC in case of Kemerer). This incorrect use of the data invalidates all the results presented in the paper. Therefore, the data needs to be cleaned (to this end look at best practise used by Sarro TOSEM2018) and all the experiments re-run.

We have clearned the datasets and re-done the experiments. Please see section3.1

R2_02: Also, the following concerns in the use of evolutionary approaches and their evaluation need to be fixed/clarified: - Multi-objective Evolutionary Algorithms produce a Pareto Front (i.e., multiple solutions/prediction models) when trained. The paper states that the median is used (med = 50th percentile of the test scores seen in the M*N results) but it remains very unclear to me how the solutions from pareto front obtained on training set are selected and used on the test set for evaluation purposes. Clarifying this point is crucial as a wrong selection can bias the results.

Good Point, We have removed the comparison with multi-objective methods since its not fair, we pay more attention on DE and FLASH's results.

R2_03: - The MOEA needs to be compared using Pareto Front quality indicators such as Hypervolume (see the ICSE16 paper by Sarro et al. for best practise on using and evaluating MOEA for effort estimation).

Good Point, We have removed the comparison with multi-objective methods since its not fair, we pay more attention on DE and FLASH's results.

R2_04: - What are the fitness functions used by DE and NSGAII?

Good Point, We have removed the comparison with multi-objective methods since its not fair. We ran our optimizers looking to improve SA (run1) or MRE (run2).

R2_05: - The settings choice for DE and NSGAII seems unfair and this makes the related argument about kilo-optimizers fictional.

Good Point, We have removed the comparison with multi-objective methods since its not fair. And we no longer say kilo-optimizer.

R2_06: Second previous approaches (including some previously proposed by the authors) for hyperparameter optimization have been neglected and not included in the empirical evaluation: - FLASH, SWAY, GALE (all proposed by the authors of this manuscript)

Good point, we have added FLASH method in our experiment for new comparison(not SWAY or GALE since SWAY superceded GALE and FLASH is our now our current high-watermark).

R2_07: - Random forest and Support Vector Regression show better results than CART and KNN for the datasets under investigation (see Sarro-TOSEM18), but they have not been included in this study, why?

Good point about RF and SVR. We have added them in our experiments.

R2_08: - Tabu Search (TS), which has been used in the past as optimizer for hyperparameter tuning for effort estimation (Corazza et al. EMSE2013 and PROMISE2010). The inclusion of TS is recommended because local search is generally simpler and faster than the global search (i.e., the MOEA used in this study) - note that the justification given by the authors to exclude TS from their analysis is not convincing, indeed those studies which have not used it in the past were not focusing on hyperparameter tuning.

This is a good point,. Our comment on TS is incomplete. We will remove.

R2_09: The novelty and contributions of the work are overclaimed: - ”hyperparameter optimization has not been extensively explored for effort estimation”. I disagree with this sentence, as there exist previous work doing proper and thorough analysis investigating hyperparameter optimization for effort estimation including the one by Corazza et al. EMSE 2013.

Good point– we need to give more pomient to the Carozza work. We have removed that that over claim in the abstract and added Carozza references to the introduction.

- Lesson 1 and first claimed contribution: previous studies already showed that off-the-shelf default should be deprecated, so lesson 1 just confirms previous findings (using even the same datasets). Please revise Lesson 1 and the first claimed contribution ”A demonstration that default settings are not the best way [...]” accordingly.

- Second claimed contribution: ”commissioning a new effort estimator requires extensive testing”, I personally do not agree with this sentence and furthermore believe the study does not provide any empirical evidence to support this claim.

R2_10: - Third claimed contribution: ”A new criteria* for assessing effort estimators”: The time required to generate the estimates is not a new criterion at all, it has been pointed out in previous work such as Sarro (TOSEM18) and Chen et al. (TSE2018). (*note criteria should be criterion).

Good point. We have deleted this part.

R2_11: The presentation quality needs major improvements: - There is a conceptual issue in the use of data miners as opposed to machine learners/prediction models, please avoid using the term data miners as this work is not about data mining.

We have removed the part using the term data miners.

- There are many typos, incomplete sentences, punctuation errors, etc. please do a careful proof-reading to bring this paper to the high-quality standard required by top scientific journals such as EMSE.

We will fix these issues.

Replies to Reviewer3 Comments

Thank you for your insightful comments, especially about our misuse of NSGA-II and MOEA/D.

On consideration, and prompted by some remarks of yours and reviewer2, we have replaced our multi-objective methods (since we do not explore multi-objectives here).

Reviewer 3: This paper studies hyperparameter optimization for effort estimation by conducting an extensive empirical evaluation with 945 projects, two learners, and seven optimizers (including DE, NSGA-II, MOEA/D). The results show that (allegedly) significant improvements can be obtained in effort estimation with proper tuning and that some combinations (CART + DE) are better than others.

While the paper might have some exciting contributions to effort estimation, it is incredibly problematic in so many dimensions:

1) The paper is full of unsupported claims and statements that reflect authors' opinions or that are incorrect due to a very unprecise and rough knowledge of the topic

2) The comparison with NSGA-II and MOEA/D is unfair (different stopping conditions for different optimizers)

3) Missing details about the parameter tuning for NSGA-II and MOEA/D

4) Missing comparison with optimizers that the same authors published in recent venues (optimizers they claimed to outperform more standard optimizers for SBSE)

5) The performance metrics adopted are not suitable for comparing multi-objective optimizers

6) How do we tune the optimizers? This paper does not solve the parameter tuning problem, but it just introduces a higher-level tuning problem.

Thanks for your comments. We will rewrite all the texts that you pointed out in your comments and add more references to support our claims. We think the new version will solve all your concerns in different regards like (1, 2, 3, 4, 5). For other concerns, please see the each response attached.

As to point (6), this paper does not explore hyper-hyper parameter optimization,since we show that with our current settings we clearly out-perform prior work.

But that is not to say that we should stop there. This draft does agree with you that hyper-hyper parameter optimziation problem is the next big issue in this field. We say so in our conclusion. Future work is required on this point.

POINT 1

R3_01: The authors should not make statements or claims that ARE NOT verifiable and do NOT reflect the related literature nor the results of this paper. These claims are described and commented below in the order of their appearance in the paper.

R3_02: ”Unlike the test case generation domains explored by Arcuri & Fraser, hyperparameter optimization for effort estimation is both useful and fast.” The results show that hyperparameter optimization is NOT fast in effort estimation as well. As stated by the authors, getting the results with the best combination of learner and optimizer ”can achieve in a few hours what other optimizers need days to weeks of CPU to accomplish.” Well, going from a few seconds (learner without optimization) to a few hours is something we cannot define as ”fast.” Is a longer running time worth?

Indeed so. Looking at Table 9 and Table 10, and comparing the best to worst results for each data set, we can see some very large improvements (e.g. albrecht MRE ATML to CART.FLASH). Note that the worst case performances come from untuned learners and the best case performances come from tuned ones. R3_03: ”Some of that prior work suffered from [] optimizing with algorithms that are not representative of the state-of-the-art in multi-objective optimization [44]”. Which of the prior work do not use representative state-of-the-art optimizers? If we go in that direction, this paper also fails to use more state-of-the-art (recent) optimizers, such as GDE3 (the generalized differential evolution) or the indicator-based optimizer, two-archive search, dimensionality-reduction based EAs, clustering-based EAs. Some of these optimizers have been published in the leading journal ”IEEE Transactions on Evolutionary Computation” in the latest issues.

We added a comment on this in sect2.2, explore all variants of all optimizers seen in all research areas is impractical. We justify the choice of our methods via an analysis of the recent SE literature.

R3_04: ”Another issue with prior work is that researchers use methods deprecated in the literature.” Based on this statement, I guess the authors have never heard of the ”no free lunch theorem” in optimization. Based on this theorem, that there are no ”deprecated optimizers” since there exist problems where even a simple grid search is effective and fast (e.g., when the search space is small in size).

Good point- we have deleted that claim

In their statement, the authors refer to ”deprecated methods” in the plural form. I want to see what is the list of optimizers that authors (by personal opinion) consider as ”deprecated.” Furthermore, the authors need to add some references from the optimization literature if they want us to believe in their (unproven) claims

You are right that this work did not draw from the current state of the art in new research research in optimization. To address that issue, we added result from a bayesian parameter optimization method (FLASH). As the reviewer knows, BPO is the current optimizer of choice for many industrial and research applications.

R3_05: ”NSGA-II is a standard genetic operator [] with a fast selection operator”. First, NSGA-II is not the standard GA, but it is a multi-objective GA, the two things are not synonyms. Second, NSGA-II does not have a fast selection operator (faster than its predecessor) but a fast algorithm to sort the population in non-dominated fronts.

Good Point, We have removed the comparison with multi-objective methods since its not fair.

R3_06: ”Kilo-optimizers” such as NSGA-II examine 103 candidates”. Actually, Kilo-optimizers do not exist, but it is a pure ”fictional” category invented by the authors to convince they did discover the holy grail for hyperparameter optimization in effort estimation. However, NSGA-II is a kilo-optimizer only because the authors use a kilo-setting for this algorithm while they use a completely different parameter setting for DE. You can run NSGA-II with a population size of 10 individual and for 8 generations. The whole argumentation about Kilo-optimizers has no sense and highlight a massive flaw in the study and the conclusions drawn (see my later comments).

Good Point, We have removed the comparison with multi-objective methods since its not fair. Plus, we reset the methods in comparison with the same budget.

POINT 2

R3_07: The biggest issue in this paper is the unfair configuration for the different optimizers: 1) The number of generations for DE is set to gen = 2, 8 (page 9, lines 42-43) 2) The number of generations for NSGA-II and MOEA/D is not specified. However, later in the paper, the authors report ”Kilo-optimizers” such as NSGA-II examine 103 candidates or more since they explore population sizes of 102 for many generations” (page 18, lines 8-9). This highlights two things: (1) NSGA-II (likely MOEA/D too) was configured with more than 2 or 8 generations. This makes the whole comparison wrong. Big surprise: DE is faster than NSGA-II because the authors run the latter longer than the former. Hence, the results do not show any superior performance of DE but simply spot an unfair setting.

Good Point, We have removed the comparison with multi-objective methods since its not fair. Plus, we reset the methods in comparison with the same budget.

R3_08: Second, are the authors aware of that DE, NSGA-II, and MEAD/D are all population-based optimizers? What is the population size used for the three algorithms? Is it the same? If not, why? I warmly recommend the authors to read the following survey by Piotrowski about the population size for DE.

We have added the description of these parameter settings. The settings we used as based on population sizes seen in prior publications.

R3_09: It seems that the authors have no idea on how to make a fair comparison among optimization algorithms. I strongly recommend the read the following reports:

[CEC18] S. Jiang et al., ”Benchmark Functions for the CEC’2018 Competition on Dynamic Multiobjective Optimization”, CEC 2018. [CEC17] R. Cheng et al., ”Benchmark Functions for CEC’2017 Competition on Evolutionary Many-Objective Optimization”, CEC 2017. [CEC15] M. Helbig et al., ”Benchmark functions for CEC 2015 special session and competition on dynamic multi-objective optimization”, CEC 2015.

If the authors carefully read the instructions for the competitions, they will find out that a fair comparison is made by configuring all algorithms with the same number of fitness evaluations.

At your suggestion, we have removed the comparison with multi-objective methods since its not fair. Plus, we reset the methods in comparison with the same budget, and re-ran the experiments.

POINT 3

R3_10: Furthermore, other important details are missing: - What are the crossover and mutation operators for NSGA-II and MOEA/D? - What are the crossover and mutation probabilities for NSGA-II and MOEA/D? - What is the number of aggregation coefficient vectors used in MOEA/D? - What is the off-the-shelf setting used for CART and ABE? Are the default ones in WEKA? The default ones reported in previous papers? The default ones in the Scikit-learn? Unclear.

We have removed the comparison with multi-objective methods since its not fair. Plus, we reset the methods in comparison with the same budget, add the details of methods parameters and re-done the experiment.

R3_11: The motivation used for not using Simulating annealing as a baseline is not sound. Simulating annealing is very fast as it uses/refines only one single solution (it is a better variant of the hill climbing). A simple baseline (with one single point to refine) must be used because (1) it has been used in the literature and we don't know if DE is better than an existing strategy, (2) SA is a better baseline for numerical problems compared to a nave random search.

In effot estimation domain, simulated is not widely used but its a good suggestion to try and it can be added to future work.

As to this point, we justify our choice of optimizers usign the following paragraph (start of Section 2.2):

> Note that we do not claim that the above represents all methods for effort estimation.Rather, we say that (a) all the above are either prominent in the literature or widely used; and (b) anyone with knowledge of the current effort estimation literature would be tempted to try some of the above

POINT 4

R3_12: It is very disappointing seeing that the authors did not compare existing optimizers with other optimizers proposed by the same authors in different venues. This is particularly true because the newly developed optimizers are claimed to be the holy grails for the whole Search-Based Software Engineering field. To be more precise, these are the optimizers: 1) FLASH from the paper "Finding Faster Configurations using FLASH." 2 500+ Times Faster Than Deep Learning, MSR 2018 3) GALE: Geometric Active Learning for Search-Based Software Engineering, TSE 2015 4) "Faster discovery of faster system configurations with spectral learning," ASE 2018 5) SWAY from the paper "Beyond evolutionary algorithms for search-based software engineering," IST 2018

Why are the optimizers mentioned above ignored? Are they better for all SBSE problem (including effort estimation)? I want to see them applied for Effort Estimation as well. If they are the best of the best, they should be included among the baselines.

Good point, we have added FLASH method in our experiment for new comparison(not SWAY or GALE since SWAY superceded GALE and FLASH is our now our current high-watermark).

POINT 5

The authors use multi-objective optimizers but fail to use proper evaluation metrics for multi-objective problems.

R3_13: First, NSGA-II and MOEA/D produce an approximation of the Pareto front (often referred as to non-dominated front), which has multiple alternative solutions. How do the authors combine the performance scores of the Pareto solutions? They "report the median (termed med) which is the 50th percentile of the test scores seen in the M*N results". How can the authors compute the median across M*N results if in each results NSGA-II (or MOEA/D) produce multiple solutions? Did the authors compute the median (per front) of the median (across the M*N results)? If this is the case, the comparison is faulty.

To compare multi-objective optimizers, there performance metrics specific for multi-objective problems, such as the IGD (Inverted Generational Distance), HV (Hypervolume), spacing, number of generated solutions that also belong to the reference front, number of solutions dominated by other optimizers.

If the authors have no idea of what these metrics are, I warmly recommend them to read (at least) carefully the CEC competition papers above.

We have removed the comparison with multi-objective methods since its not fair. Plus, we reset the methods in comparison with the same budget.

POINT 6

R3_14: This paper does not solve the parameter setting problem, but it merely shifts it at a higher level. Instead of tuning the parameters for the learners, we have to tune the parameters of the optimizers. This aspect must be discussed in (1) introduction, (2) threats to validity and (3) conclusions.

Why did the authors use the optimizers (e.g., DE) with their off-the-shelf settings? Notice, we cannot use as argumentation the paper by Arcuri and Fraser as the authors of this manuscript challenge them for the learners. Could it be that Arcuri and Fraser's results apply to evolutionary algorithms and not to learners? Here, we need some clarifications from the authors.

As said above, this paper does not explore hyper-hyper parameter optimization,since we show that with our current settings we clearly out-perform prior work.But that is not to say that we should stop there. This draft does agree with you that hyper-hyper parameter optimziation problem is the next big issue in this field.We say so in our conclusion. Future work is required on this point

R3_15: FURTHER COMMENTS: In page 11, do we need 6 lines of text to explain a simple 10-folds cross-validation?

R3_16: Readers of EMSE know what the Scott-Knott test is. There is no need for Table 6. The extra page could be used to provide more critical information (as pointed out in my previous comments).

We added these description to help those out-of-domain readers to understand easier.

R3_17: Typos and mistakes: Page 1, line 21: such hyperparameter -¿ such a hyperparameter Page 7, the parenthesis opened in line 46 is never closed Page 8, the parenthesis opened in line 41 is never closed Page 15, broken sentence in lines 47-48

Good catch, we will fixed them.