Exploiting the Essential Assumptions of Analogy-based Effort Estimation

Ekrem Kocaguneli, *Student Member, IEEE,* Tim Menzies, *Member, IEEE,* Ayse Bener, *Member, IEEE,* and Jacky W. Keung, *Member, IEEE*

Abstract—
Background: There are too many design options for software effort estimators. How can we best explore them all?
Aim: We seek aspects on general principles of effort estimation that can guide the design of effort estimators.
Method: We identified the essential assumption of analogy-based effort estimation: i.e. the immediate neighbors of a project offer stable conclusions about that project. We test that assumption by generating a binary tree of clusters of effort data and comparing the variance of super-trees vs smaller sub-trees.
Results: For ten data sets (from Coc81, Nasa93, Desharnais, Albrecht, ISBSG, and data from Turkish companies), we found: (a) the estimation variance of cluster sub-trees is usually *larger* than that of cluster super-trees; (b) if analogy is restricted to the cluster trees with lower variance then effort estimates have a significantly lower error (measured using MRE and a Wilcoxon test, 95% confidence, compared to nearest-neighbor methods that use neighborhoods of a fixed size).
Conclusion: Estimation by analogy can be significantly improved by a dynamic selection of nearest neighbors, using only the project data from regions with small variance.

Index Terms—Software Cost Estimation, Analogy, k-NN

1 Introduction

Software effort estimates are often wrong by a factor of four [?] or even more [?]. As a result, the allocated funds may be inadequate to develop the required project. In the worst case, over-running projects are canceled and the entire development effort is wasted. For example:

- NASA canceled its incomplete Check-out Launch Control System project after the initial \$200M estimate was exceeded by another \$200M [?].
- The ballooning software costs of JPL's Mission Science Laboratory recently forced a two-year delay [?].

It is clear that we need better ways to generate project effort estimates. However, it is not clear how to do that. For example, later in this paper we document nearly thirteen thousand variations for analogy-based effort estimation (ABE). Effort estimation is an active area of research [?], [?], [?], [?] and more variations are constantly being developed. We expect many more variations of ABE, and other effort estimation methods, to appear in the very near future.

Recent publications propose data mining toolkits for automatically exploring this very large (and growing) space of options for generating effort estimates. For example, in 2006, Auer et al. [?] propose an extensive search to learn the best weights to assign different project features. Also in that year, Menzies et al. [?]'s COSEEKMO tool explored thousands of combinations of discretizers, data pre-processors, feature subset selectors, and inductive learners. In 2007, Baker proposed an exhaustive search of all possible project features, learners, etc. He concluded that such an exhaustive search was impractical [?].

The premise of this paper is that we can do better than a COSEEKMO-style brute-force search through the space of all variants of effort estimators. Such studies are computationally intensive (the COSEEKMO experiments took two days to terminate). With the ready availability of cheap CPU farms and cloud computing, such CPU-investigations are becoming more feasible. On the other hand, datasets containing historical examples of project effort are typically small[1]. In our view, it seems misdirected to spend days of CPU time just to analyze a few dozen examples. These CPU-intensive searches can generate gigabytes of data. Important general properties of the estimation process might be missed, buried in all that data. As shown below, if we exploit these aspects, we can significantly improve effort estimates.

One alternative to the brute-force search of COSEEKMO is heuristic search. For example Li et al. [?] do not explore all options of their estimators. Rather, they use a genetic algorithm to guide the search for the best project features and the best cases to be

- *Ekrem Kocaguneli is with the Department of Computer Engineering, Bogazici University. E-mail: ekrem.kocaguneli@boun.edu.tr*
- *Tim Menzies is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us*
- *Ayse Bener is with the Department of Computer Engineering, Bogazici University. E-mail: bener@boun.edu.tr*
- *Jacky W. Keung is with the School of Computer Science and Engineering, University of New South Wales. E-mail: jacky.keung@nicta.com.au*

This research is funded in part by Tubitak EEEAG108E014.

1. For example, the effort estimation datasets used in Mendes et al. [?], Auer et al. [?], Baker [?], this study, and Li et al. [?] have median size (13,15,31,33,52), respectively.

used in the training data. However, while guided search might comment on *which* particular variant is best for some data set, it will not comment on *why* that variant is best. Hence, they do not offer general principles that could simplify future effort estimation work.

This paper proposes an alternative to brute-force and heuristic search. According to our *easy path* principle for designing an effort predictor:

Find the situations that confuse estimation. Remove those situations.

The easy path is not standard practice. Usually, prediction systems are matured by adding mechanisms to handle the harder cases. For example, the AdaBoost algorithm generates a list of learners, and each learner focuses on the examples that were poorly handled by the one before [**?**].

Focusing on just the easy cases could be problematic. If we only explore the easy cases, we could perform badly on the hard test cases. On the other hand, if the easy path works, it finds short-cuts that simplifies future effort estimation work. Also, it avoids COSEEKMO's brute-force search since, according to this principle, we only explore the options that challenge the essential assumptions of the predictor.

The rest of this paper uses the easy path to build and evaluate an effort estimator called TEAK (short for "Test Essential Assumption Knowledge"). In keeping with the easy path, we only explored design options that commented on TEAK's essential assumptions; specifically: (a) case subset selection and (b) how many training examples should be used for estimation.

TEAK's design applied the easy path in five steps:

1) Select a prediction system.
2) Identify the predictor's essential assumption(s).
3) Recognize when those assumption(s) are violated.
4) Remove those situations.
5) Execute the modified prediction system.

On evaluation, we found that for the data sets studied here, TEAK generated significantly *better* estimates than comparable methods.

More generally, the success of the easy path principle recommends it for future research. When designing a predictor, it is useful to *first* try optimizing for the situations where prediction is easy, *before* struggling with arcane and complex mechanisms to handle the harder situations. For example, in future work, we will apply steps 1,2,3,4,5 to other aspects of effort estimation like feature weighting, and similarity measures.

The rest of this paper is structured as follows. After a review of the general field of effort estimation, we will focus on ABE (analogy-based estimation). For ABE, we will work through the above five steps to design TEAK. TEAK's performance will then be compared against six other ABE systems. Our conclusion will be to recommend TEAK for effort estimation.

2 Background

After Shepperd [?], we say that software project effort estimation usually uses one of three methods:

- Human-centric techniques (a.k.a. expert judgment);
- Model-based techniques including:
  - Algorithmic/parametric models such COCOMO [?], [**?**];
  - Induced prediction systems.

Human centric techniques are the most widely-used estimation method [**?**], but are problematic. If an estimate is disputed, it can be difficult to reconcile competing human intuitions (e.g.) when one estimate is generated by a manager who is senior to the other estimator. Also, Jorgensen [**?**] reports that humans are surprisingly poor at reflecting and improving on their expert judgments.

One alternative to expert judgment is a model-based estimate. Models are a reproducible methods for generating an estimate. This is needed for (e.g.) U.S. government software contracts that require a model-based estimate at each project milestone [?]. Such models are used to generate and audit an estimate, or to double-check a human-centric estimate.

Model-based estimates can be generated using an algorithmic/parametric approach or via induced prediction systems. In the former, an expert proposes a general model, then domain data is used to tune that model to specific projects. For example, Boehm's 1981 COCOMO model [?] hypothesized that development effort was exponential on LOC and linear on 15 *effort multipliers* such as analyst capability, product complexity, etc. Boehm defined a *local calibration* procedure to tune the COCOMO model to local data.

Induced prediction systems are useful if the available local training data does not conform to the requirements of a pre-defined algorithmic/parametric model such as COCOMO. There are many induction methods including stepwise regression, linear regression, rule induction, neural nets, model trees, and analogy, just to name a few [**?**], [?]. All these methods have underlying assumptions. For example, linear regression assumes that the effort data fits a straight line while model trees assumes that the data fits a set of straight lines. When data violates these assumptions, various patches have been proposed. Boehm [?, p526-529] and Kitchenham & Mendes [**?**] advocate taking the logarithms of exponential distributions before applying linear regression. Selecting the right patch is typically a manual process requiring an analyst experienced in effort estimation. TEAK, on the other hand, is a fully automatic process requiring no expert control. When experts like Boehm, Kitchenham, or Mendes are available, patching can significantly improve effort estimation. Otherwise, automatic methods should be considered.

2.1 Analogy-based Estimation (ABE)

In ABE, effort estimates are generated for a *test* project by finding similar completed software projects (a.k.a.

the *training projects*). Following Kadoda & Shepperd [?], Mendes et al. [?], and Li et al. [?] we define a baseline ABE called ABE0, as follows.

ABE0 executes over a table of data where:

- Each row is one project;
- Columns are either *independent* features seen in the projects or one special *dependent* fortune that stores the effort required to complete one project.

After processing the training projects, ABE0 inputs one test project then outputs an estimate for that project. To generate that estimate, a *scaling measure* is used to ensure all independent features have the same degree of influence on the distance measure between test and training projects. Also, a *feature weighting* scheme is applied to remove the influence of the less informative independent features. For example, in feature subset selection [?], some features are multiplied by zero to remove redundant or noisy features.

After feature scaling and weighting, ABE0 computes the distance between the test and training via a *a similarity measure* that combines all the independent features. ABE0 uses the standard Euclidean measure to assess the distance between two projects x and y with normalized independent features x_i and y_i

$$Distance = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2} \tag{1}$$

In the above, w_i is the feature weighting. ABE0 uses a uniform weighting scheme; i.e. $w_i = 1$.

Once similarity can be calculated, it must be determined *how many analogies* (i.e. subsets of the training projects) will be used for estimation. Once the k-th nearest analogies are found, they must be *adapted* to generate an effort estimate. Adaption need not be a complex task: ABE0 will return the median estimate of the k nearest analogies.

2.2 Alternatives to ABE0

The literature lists many variants for ABE0-like systems. A sample of those variants are discussed below.

2.2.1 Three Case Subset Selectors

A *case subset selection* is sometimes applied to improve the set of training projects. These selection mechanisms are characterized by how many cases they remove:

- *Remove nothing:* Usually, effort estimators use all training projects [?], [?]. ABE0 is using this variant.
- *Outlier* methods prune training projects with (say) suspiciously large values [?]. Typically, this removes a small percentage of the training data.
- *Prototype* methods find, or generate, a set of representative examples that replace the training cases. Typically, prototype generation removes most of the training data. For example, Chang's prototype generators [?] replaced training sets of size $T = (514, 150, 66)$ with prototypes of size $N = (34, 14, 6)$ (respectively). That is, prototypes may be as few as $\frac{N}{T} = (7, 9, 9)\%$ of the original data.

For example, our reading of Li et al. [?] is that their genetic algorithms are more an *outlier* technique than a *prototype* technique.

2.2.2 Eight Feature Weighting Methods and Five Discretizers

In other work Keung [?], Li et al. [?], and Hall & Holmes [?] review eight difference feature weighting schemes. Closer inspection reveals many more design variants of feature weighting. Some feature weighting schemes require an initial *discretization* of continuous columns. Discretization divides a continuous range at break points $b_1, b_2, ...$, each containing a count $c_1, c_2, ...$ of numbers [?]. There are many discretization policies in the literature including:

- Equal-frequency, where $c_i = c_j$;
- Equal-width, where $b_{i+1} - b_i$ is a constant;
- Entropy [?];
- PKID [?];
- Do nothing at all.

2.2.3 Three Similarity Measures

Mendes et al. [?] discuss three similarity measures [?], [?] including the Euclidean measure described above and a "maximum distance" measure that that focuses on the single feature that maximizes inter-project distance.

2.2.4 Six Adaption Mechanisms

With regards to adaptation, the literature reports many approaches including:

- Report the median effort value of the analogies;
- Report the mean dependent value;
- Report a weighted mean where the nearer analogies are weighted higher than those further away [?];
- Summarize the adaptations via a second learner; e.g. regression [?], model trees [?], [?] or neural network [?].

2.2.5 Six Ways to Select Analogies

Li et al. [?] comment that there is much discussion in the literature regarding the number of analogies to be used for estimation. Numerous methods are proposed, which we divide into *fixed* and *dynamic*.

Fixed methods use the same number of analogies for all items in the test set. For example, Li et al. [?] report that a standard fixed method is to always use $1 \leq k \leq 5$ nearest projects:

- $k = 1$ is used by Lipowezky et al. [?] and Walkerden & Jeffery [?];
- $k = 2$ is used by Kirsopp & Shepperd [?]
- $k = 1, 2, 3$ is used by Mendes el al. [?]

Dynamic methods adjust the number of analogies, according to the task at hand. For example, following

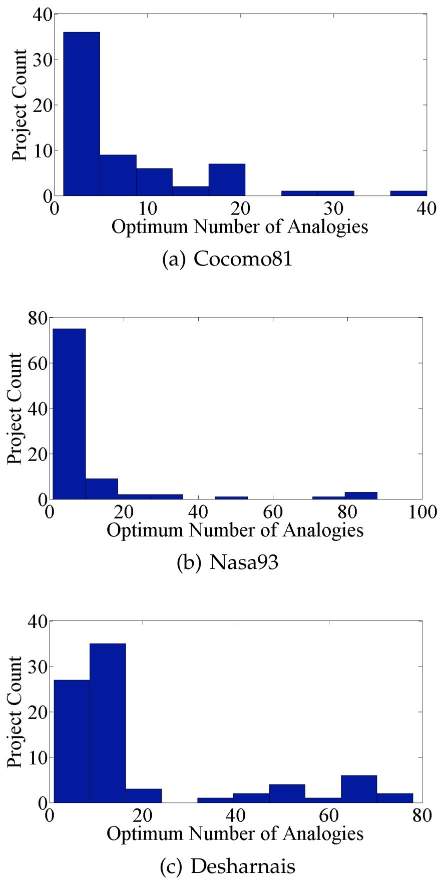


Fig. 1: Distribution of k after removing each project instance, then applying Best(K) on the remaining data. The y-axis counts the number of times a particular k value was found by Best(K).

advice from Martin Shepperd[2], Baker [?] tuned k to a particular training set using the following "Best(K)" procedure:

1) Select $N \subseteq T$ training projects at random;
2) For each $k \in 1..T-N$, compute estimates for $n \in N$;
3) Find the k value with least error in step 2.
4) When estimating, use the k-nearest neighbors, where k is set by step 3.

Our results show that Best(K) out-performs the standard fixed methods (i.e. $k \leq 5$). More interestingly, as shown in Figure **??**, Best(K) recommends k values that are very different to those seen in the standard fixed methods. These results come from three commonly used data sets (Desharnais, NASA93, and the original COCOMO data set from [?]: for notes on these data sets, see the appendix).

While ABE systems differ on many apsects, they all use analogy selection . The Figure **??** results suggest that there may be something sub-optimal about standard, widely-used, fixed selection methods. Hence, the rest of this paper takes a closer look at this aspect of ABE.

2. Personal communication.

3 Designing TEAK

The above sample of the literature, describes

$$3 * 6 * 8 * 5 * 6 * 3 = 12,960$$

ways to implement similarity, adaptation, weighting, etc. How might we explore all these variations?

The rest of this paper applies the easy path to design and evaluate an ABE system called TEAK (Test Essential and Assumption Knowledge). TEAK is an ABE0, with the variations described below.

3.1 Select a Prediction System

Firstly, we *select a prediction system*. We use ABE since:

- It is a widely studied [?], [?], [**?**], [?], [**?**], [?], [**?**], [**?**], [**?**], [**?**], [?], [**?**], [?], [**?**].
- It works even if the domain data is sparse [**?**].
- Unlike other predictors, it makes no assumptions about data distributions or an underlying model.
- When the local data does not support standard algorithmic/parametric models like COCOMO, ABE can still be applied.

The easy path limits the space of design options to just *those that directly address the essential assumptions of the predictor*. As shown below, for ABE, this directs us to issues of case subset selection and the number of analogies used for estimation.

3.2 Identify Essential Assumption(s)

The second step is to *identify the essential assumptions of that prediction system*. Although it is usually unstated, the basic hypothesis underlying the use of analogy-based estimation is that projects that are similar with respect to project and product factors will be similar with respect to project effort [?]. In other words:

Assumption One: Locality implies uniformity.

This assumption holds for project training data with the following property:

- The K-nearest training projects with effort values $E_1, E_2, .., E_k$ have a mean value $\mu = \left(\sum_i^k E_i\right)/k$ and a variance $\sigma^2 = \left(\sum_i^k (E_i - \mu)^2\right)/(k-1)$.
- By $Assumption\ One$, decreasing k also decreases σ^2.

3.3 Identify Assumption Violation

The third step is to *recognize situations that violate the essential assumption*. Implementing this step requires some way to compare the variance of larger-k estimates to smaller-k estimates. We will use Greedy Agglomerative Clustering (GAC) and the distance measure of Equation **??**. GAC is used in various fields (data mining [**?**], databases [**?**] bioinformatics [**?**]). GAC executes bottom-up by grouping together at a higher level $(i+1)$ the closest pairs found at level i. The result is a tree like Figure **??**. GAC is "greedy" in that it does not pause to

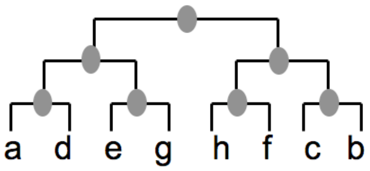


Fig. 2: A GAC tree built from 8 projects. Leaf vertices are actual projects. Grey vertices are the median of their children. Only the leaf nodes of this tree hold effort data from historical projects.

consider optimal pairings for vertices with very similar distances.

Using a GAC tree, finding the k-nearest neighbors in project data can be implemented using the following TRAVERSE procedure:

1) Place the test project at the root of the GAC tree.
2) Move the test project to the nearest child (where "nearest" is defined by Equation **??**).
3) Go to step 2

Clearly, a $k = 1$ nearest-neighbor estimate comes from TRAVERSE-ing to a leaf, then reporting the effort of that leaf. More generally, a $k = N$ nearest-neighbor estimate comes from TRAVERSE-ing to a sub-tree with N leaves, then reporting the median efforts of those leaves.

TRAVERSE can test *Assumption One*. Let some current vertex x have children y and z. We say that:

- The sub-trees starting at x, y, z have leaves L_x, L_y, L_z (and $L_x = L_y \cup L_z$).
- The number of sub-tree leaves is $k_x = k_y + k_Z$.
- The variance of the leaves' efforts are $\sigma_x^2, \sigma_y^2, \sigma_z^2$.
- After C4.5 [**?**], we say the variance of the trees below x (denoted σ_{yz}^2) is the weighted sum:

$$\sigma_{yz}^2 = \frac{k_y}{k_x}\sigma_y^2 + \frac{k_z}{k_x}\sigma_z^2$$

Parent trees have the nodes of their children (plus one). If we TRAVERSE from a parent x to a child, then the sub-tree size k decreases. That is, TRAVERSE-ing moves into progressively smaller sub-trees.

Assumption One holds if, when TRAVERSE-ing from all vertices x with children y and z, the sub-tree variance *decreases*. That is:

$$\forall x \in T : \sigma_x^2 > \sigma_{yz}^2 \tag{2}$$

3.4 Remove Violations

The fourth step in TEAK's design is to *remove the situations that violate the essential assumption*. We instrumented TRAVERSE to report examples where Equation **??** was violated; i.e. where it recursed into sub-trees with a larger variance than the parent tree. We found that this usually occurs if a super-tree contains mostly similar effort values, but one sub-tree has a minority of outliers. For example:

- Suppose some vertex x has children y, z.
- Let each child start sub-trees whose leaves contain the effort values $leaves(y) \in \{1253, 1440\}$ staff hours and $leaves(z) \in \{1562, 5727\}$ staff hours.

In this example:

- The leaves of the parent tree x have similar effort values: 1,253 and 1,562 and 1,440 staff hours.
- But the leaves of the subtree z has outlier values; i.e. 5,727.
- TRAVERSE-ing from the super-tree x to the sub-tree z increases the variance by two orders of magnitude.

A *sub-tree pruning policy* is used to prune sub-trees with a variance that violates the essential assumption. We experimented with various policies that removed subtrees if they had:

1) more than α times the parent variance;
2) more than $\beta * max(\sigma^2)$;
3) more than $R^\gamma * max(\sigma^2)$, where R is a random number $0 \leq R \leq 1$.

In order to avoid over-fitting, our pruning policy experiments were restricted to one data set (Boehm's COCOMO embedded projects [?]) then applied, without modification, to the others. The randomized policy (#3) produced lowest errors, with smallest variance. The success of this randomized policy suggests two properties of effort estimation training data:

- The boundary of "good" training projects is not precise. Hence, it is useful to sometimes permit random selection of projects either side of the boundary.
- The policy tuning experiments recommended $\gamma = 9$. This selects for subtrees with less than 10% of the maximum variance[3]. This, in turn, suggests that the above example is typical of effort estimates; i.e. subtree outliers are usually a few large effort values.

3.5 Execute the Modified System

The final step in the design of TEAK is to build a new prediction system. TEAK executes as follows:

- Apply GAC to the training projects to build a tree called GAC1;
- Prune GAC1 using the sub-tree pruning policy described above. The remaining leaves are the *prototypes* to be used in effort estimation.
- Apply GAC to the prototypes to build a second tree called GAC2.
- Place the test project at the root of GAC2. Compute an estimate from the median value of the GAC2 projects found by TRAVERSE2. TRAVERSE2 is a variant of TRAVERSE that ensures the essential assumption is never violated. It stops recursing into GAC2 sub-trees when Equation **??** is violated.

In theory, TEAK is slow. Given T training projects, TEAK builds two GAC trees of maximum height $log_2(T)$. At each level, distances are computed between all pairs of vertices. Since the number of vertices is halved at each

3. The mean of $rand()^9 \approx 0.1$.

				Historical Effort Data					
Dataset	Features	$T = \lvert Projects \rvert$	Content	Units	Min	Median	Mean	Max	Skewness
Cocomo81	17	63	NASA projects	months	6	98	683	11400	4.4
Cocomo81e	17	28	Cocomo81 embedded projects	months	9	354	1153	11400	3.4
Cocomo81o	17	24	Cocomo81 organic projects	months	6	46	60	240	1.7
Nasa93	17	93	NASA projects	months	8	252	624	8211	4.2
Nasa93c2	17	37	Nasa93 projects from center 2	months	8	82	223	1350	2.4
Nasa93c5	17	40	Nasa93 projects from center 5	months	72	571	1011	8211	3.4
Desharnais	12	81	Canadian software projects	hours	546	3647	5046	23940	2.0
SDR	22	24	Turkish software projects	months	2	12	32	342	3.9
Albrecht	7	24	Projects from IBM	months	1	12	22	105	2.2
ISBSG-Banking	14	29	Banking projects of ISBSG	hours	662	2355	5357	36046	2.6
		Total: 448							

Fig. 3: The 448 projects used in this study come from 10 data sets. Indentation in column one denotes a dataset that is a subset of another dataset. For notes on this data, see the appendix.

next level, building two GAC trees requires the following number of distance calculations:

$$2 * \left(\sum_{i}^{log_2(T)} \left(\frac{T}{2^{i-1}} \right)^2 \right) = \frac{8}{3} \left(T^2 - 1\right)$$

This $O(T^2)$ computation is deprecated for large T. However, for this study, TEAK generates an estimate in less than a second[4]. Our runtimes were fast because GAC1 and GAC2 are usually small: Figure **??** shows all our training projects contain less than 100 examples.

4 Comparisons

Recall the pre-experimental concern expressed above: Taking the easy path might ignore important design issues, to the detriment of the predictions. To address that concern, this section compares TEAK to a range of other ABE0 variants.

4.1 Randomized Trials

When assessing TEAK, is it important to execute it using random orderings of the project data. Different orderings of the training projects in different runs can produce different trees when the greedy search of GAC selects different sub-trees violating *Assumption One*. A repeated randomization study 20 times lets us check for conclusion stability across different GAC1 and GAC2 trees:

- Twenty times, for each data set, we randomize the order of the rows in that data set.
- Next, we conduct a Leave-One-Out study; i.e. given T projects, then $\forall t \in T$, use t as the test project and the remaining $T-1$ projects for training. For these studies, we used all independent features when computing similarities.

We applied this randomized trial procedure using the 448 projects from the 10 data sets of Figure **??** (for notes on this data, see the appendix). In all, these randomized trials generated 20*448=8,960 training/test set pairs.

4. Using Matlab on a standard Intel x86 dual core notebook running LINUX with 4GB of ram.

```
$win_i$ = 0, $tie_i$ = 0, $loss_i$ = 0
$win_j$ = 0, $tie_j$ = 0, $loss_j$ = 0
if WILCOXON($MRE's_i$, $MRE's_i$) says they are the same then
    $tie_i$ = $tie_i$ + 1;
else
    if median($MRE's_i$) $<$ median($MRE's_j$) then
        $win_i$ = $win_i$ + 1
        $loss_j$ = $loss_j$ + 1
    else
        $win_j$ = $win_j$ + 1
        $loss_i$ = $loss_i$ + 1
    end if
end if
```

Fig. 4: Pseudocode for Win-Tie-Loss Calculation Between Variant i and j

4.2 Details

For each of these 8,960 pairs of training/test, estimates were generated by TEAK and six other ABE0 variants:

- Five variants returned the median effort seen in the k-th nearest neighbors for $k \in \{1, 2, 4, 8, 16\}$.
- The other variant returned the median effort seen in k neighbors found using Baker's Best(K) procedure. From §**??**, recall that Best(K) adjusts k to each data set by reflecting over all the training projects.

Since this is paired data, we applied a Wilcoxon signed rank test (95% confidence) to rank the resulting estimates. Our performance metric is the magnitude of the relative error, or MRE:

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \tag{3}$$

In addition to MRE, we also used win-tie-loss values to summarize the Wilcoxon comparisons. Each data set generated 20*7=140 MRE values for each ABE0 variant and each iteration of the randomized trials. To calculate the win-tie-loss values, we first checked if two variants i, j are statistically different according to the Wilcoxon test. If not, then we incremented tie_i and tie_j. On the other hand, if they turned out to be different, we updated win_i, win_j and $loss_i, loss_j$ after a numerical comparison of their median values. The pseudocode for win-tie-loss calculation is given in Figure **??**.

4.3 Results

The resulting Win/Loss/Ties values are shown in Figure **??**. In all ten data sets, TEAK had the largest

Data set	Variant	Win	Tie	Loss	Win - Loss
Cocomo81	TEAK	54	66	0	54
	Best(K)	26	90	4	22
	k=8	17	85	18	-1
	k=16	21	77	22	-1
	k=1	14	74	32	-18
	k=4	10	80	30	-20
	k=2	8	68	44	-36
Cocomo81e	TEAK	55	65	0	55
	Best(K)	36	81	3	33
	k=16	22	89	9	13
	k=8	17	84	19	-2
	k=4	13	83	24	-11
	k=1	10	67	43	-33
	k=2	3	59	58	-55
Cocomo81o	TEAK	11	109	0	11
	k=16	6	114	0	6
	k=8	4	116	0	4
	Best(K)	4	116	0	4
	k=4	4	114	2	2
	k=2	1	111	8	-7
	k=1	1	98	21	-20
Nasa93	TEAK	40	80	0	40
	Best(K)	22	96	2	20
	k=16	19	97	4	15
	k=8	20	91	9	11
	k=4	14	89	17	-3
	k=2	9	66	45	-36
	k=1	6	61	53	-47
NASA93c2	TEAK	21	99	0	21
	k=8	14	104	2	12
	Best(K)	14	103	3	11
	k=16	13	103	4	9
	k=4	9	100	11	-2
	k=1	8	88	24	-16
	k=2	1	83	36	-35
Nasa93c5	TEAK	24	96	0	24
	k=16	16	104	0	16
	Best(K)	14	105	1	13
	k=8	13	104	3	10
	k=4	6	100	14	-8
	k=2	3	91	26	-23
	k=1	1	86	33	-32
Desharnis	TEAK	32	88	0	32
	k=16	17	100	3	14
	k=8	15	101	4	11
	Best(K)	15	101	4	11
	k=4	16	96	8	8
	k=2	7	78	35	-28
	k=1	1	70	49	-48
SDR	TEAK	51	69	0	51
	k=8	13	98	9	4
	Best(K)	11	99	10	1
	k=4	8	101	11	-3
	k=16	9	94	17	-8
	k=1	10	88	22	-12
	k=2	1	85	34	-33
Albrecht	TEAK	3	117	0	3
	k=16	3	117	0	3
	k=8	2	118	0	2
	k=4	0	120	0	0
	Best(K)	0	120	0	0
	k=2	0	117	3	-3
	k=1	0	115	5	-5
ISBSG-Banking	TEAK	27	93	0	27
	k=16	22	98	0	22
	k=8	18	98	4	14
	Best(K)	17	100	3	14
	k=4	9	97	14	-5
	k=2	4	77	39	-35
	k=1	3	77	40	-37

Fig. 5: MRE win-loss-tie results from the *randomized assessment* procedure. Results sorted by data set, then wins minus loss. Gray cells indicate variants with zero losses.

$win - loss$ values; i.e. TEAK always achieves statistically significant lower MRE values of all the methods in this study. Also, TEAK has a $loss$ value of 0 for all datasets; i.e. TEAK was never significantly outperformed by other variants.

As to the other ABE variants, three results deserve some attention. Firstly, as mentioned above, Baker's Best(K) dynamic k selection algorithm performed better than other variants using standard fixed k sizes. This suggests that many prior results in ABE might be improved using a different analogy selection policy. Nevertheless, we do not recommend Best(K) since it only has zero losses in two cases; i.e. Best(K) is a comparatively worse method than TEAK. That it is, tuning k to entire dataset (as done with Best(K)) is less useful than tuning k to carefully selected prototypes (as done with TEAK).

Secondly, after TEAK, the method with the next fewest losses was $k = 16$. This result should interest advocates of $k \in \{1, 2, 3, 4, 5\}$ (e.g. [?]). This suggests that many current ABE methods explore too small a neighborhood.

Finally, a very interesting feature of these results is the large number of ties seen in all datasets. For example, in some data sets like Nasa93c2, Cocomo81e and Albrecht, most of the methods tied 80% of the time or more. Figure **??** explores these ties in more detail:

- Each line in that figure shows the sorted MREs for one variant on the dataset.
- On the left-hand-side of the results for each dataset, the curves are nearly identical.
- Only on the right-hand-sides, that show how badly the methods perform, do these ABE0 variants differentiate themselves.
- In those right-hand-regions, TEAK always has the lowest distribution of errors.

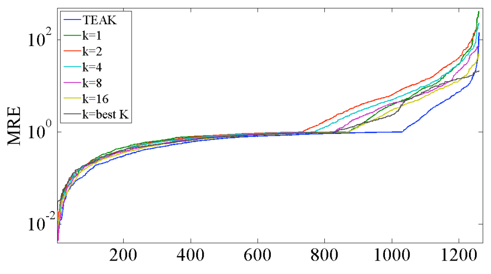
The low error distributions of TEAK in Figure **??** is more impressive that it might first appear. The y-axis of these plots is an exponential scale. That is, when estimate errors grow large, TEAK has exponentially less errors than other ABE0 variants.

In summary, the pre-experimental concern that the easy path was too simplistic was not observed. Of all the variants studied here, TEAK is unequivocally the superior ABE system.

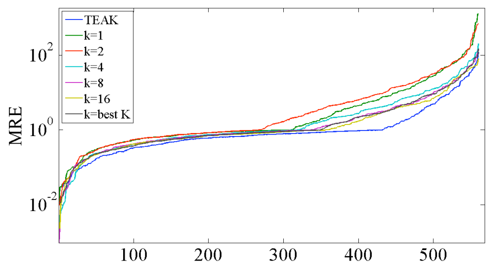
4.4 Threats to Validity

Internal validity questions to what extent the cause-effect relationship between dependent and independent variables hold [?].

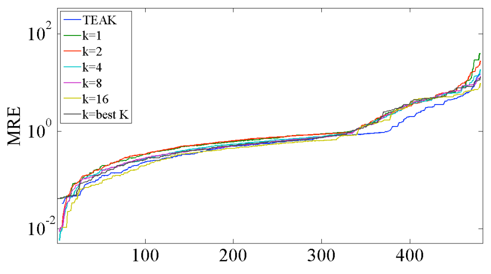
The general internal validity issue is that data mining experiments (like those discussed above) do not collect new data, but only generates theories from historical data. Ideally, we should take a learned theory and apply it to some new situation, then observe if the predicted effect occurs in practice. Note that if no explicit theory is generated, then it cannot be be applied outside of the learning system. That is, all ABE systems suffer from issues of internal validity since they do not generate an



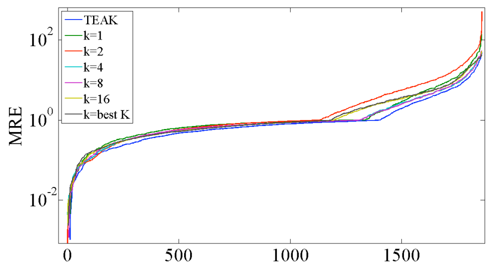
(a) Cocomo81



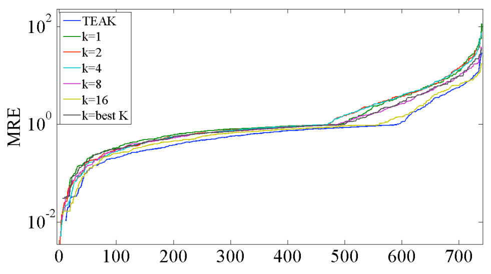
(b) Cocomo81e



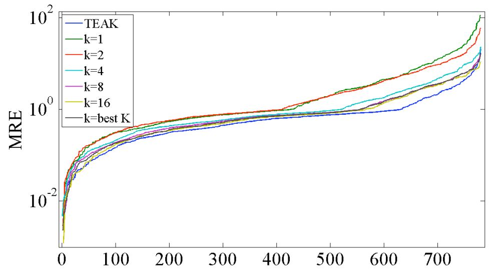
(c) cocomo81o



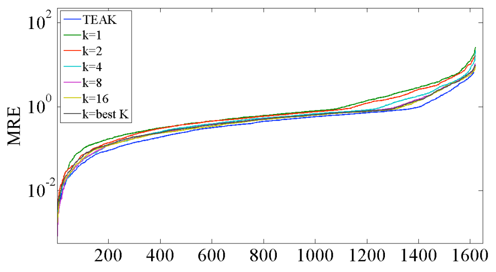
(d) Nasa93



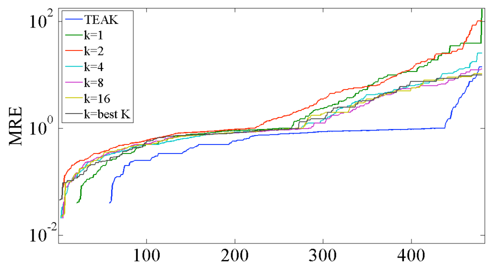
(e) Nasa93c2



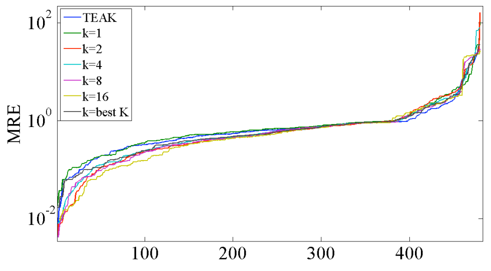
(f) Nasa93c5



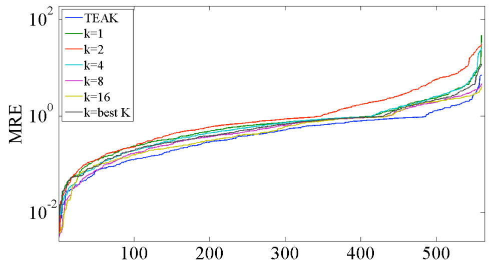
(g) Desharnais



(h) SDR



(i) Albrecht



(j) ISBSG

Fig. 6: Log-Mre Values of Datasets for GAC2 and multiple *k* Values. For each dataset, log of MRE values for 20 repeats are sorted.

explicit theory. However, it is possible to mitigate this problem by simulating how an ABE system might be applied to a new situation. Note that the Leave-One-Out approach used in this paper generates estimates using test data that is not used in training.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [?]. In our research we are using MRE and win-tie-loss values for measuring and comparing performance of different models. Another internal validity issue is our use of MRE (Magnitude of Relative Error). This is the most widely used evaluation criterion for assessing the performance of competing software effort estimation models [?], [?], [?]. Each individual project case's MRE value is a direct measure of the absolute difference between the prediction and the known actual value [?]. Therefore the smaller the MRE indicates the better the performance of the prediction system [?]. Foss et al. [?] have provided an extensive discussion demonstrating that by using only MRE itself may be leading to incorrect evaluation. This is largely due to measuring the error ratio between the error and the actual value, where the error is divided by the actual, and this will lead to an incorrect result in some circumstances. For example:

- If project case A has a predicted effort of 50 and actual effort of 100, the absolute error is 50 and the MRE value for case A is 0.5.
- In contrast, if project case B has a predicted effort of 100 and actual effort of 50, the absolute error is 50, but the MRE value for case B is 1.0.

This is an algorithmic limitation of MRE and can be resolved by applying an additional statistical evaluation, we have used Wilcoxon statistics to ensure the our error measure do not fall into the MRE limitation. i.e. if the two variants are statistically different. In practice, the probability of the aforementioned error to occur is insignificant, and will not have a major impact to the overall evaluation. Therefore, given MRE is the most commonly used method in the cost estimation research community, we use MRE to present our results.

External validity is the ability to generalize results outside the specifications of that study [?]. To ensure the generalizability of our results, we studied a large number of projects. Our datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. For example, we used datasets composed of software development projects from different organizations around the world to generalize our results [?]. Our reading of the literature is that this study uses more project data, from more sources, than numerous other papers. All the papers we have read, as well as, Table 4 of [?] list the total number of projects in all data sets used by other studies. The median value of that sample is 186, which is less than half the 448 projects used in our study.

5 Conclusion

In response to the growing number of options for designing software project effort estimators, various researchers (e.g. [?], [?], [?]) have proposed elaborate and CPU-intensive search tools for selecting the best set of design options for some local data. While useful, these tools offer no insight into the effort estimation task: they report *what* what the design is in simplifying future effort estimation tasks, but not *why* they were useful. Such insights are useful for reducing the complexity of future effort estimations.

In order to avoid the computation cost of these tools, and to find the insights that simplify effort estimation, we design TEAK using an *easy path* principle. The easy path has five steps.

1. Select a prediction system: Analogy-based effort estimation, or ABE, is a widely-studied method that works on sparse data sets. Hence, we selected ABE as our prediction system.

2. Identify the predictor's essential assumption(s): The essential assumption of ABE is that *locality implies uniformity*; i.e. the closer the test project approaches the training projects, the smaller the variance in that neighborhood.

3. Recognize when those assumption(s) are violated: Mathematically, this can be tested by recursively clustering project data into a tree whose leaves contain historical effort data and whose internal nodes are medians of pairs of child nodes. When descending this tree, the essential ABE assumption is violated when sub-trees have a larger variance than the parents.

4. Remove those situations: This assumptions can be removed by pruning sub-trees with the larger variances.

5. Execute the modified prediction system: TEAK builds a second tree of clusters using just the projects not found in high variance sub-trees. Estimates are generated from this second tree by a recursive descent algorithm that stops before the sub-tree variance is higher than the super-tree variance. The leaves of terminating sub-tree are then accessed and the estimate is calculated from the median of the effort values in those leaves.

A pre-experimental concern with the easy path was that, in ignoring the hard training cases, we would miss important aspects of the data. Our experiments do not support that concern. TEAK never lost against other ABE methods and always won the most.

We conclude that is may be detrimental to prediction to obsess on the hard cases. Rather, it may be better to enhance what a predictor does best, rather than try to patch what it does worst. For example, in the case of ABE, case selection via variance significantly improved the estimates.

6 Future Work

In this paper, we have applied the easy path principle to design a new method for case & analogy selection. In future work, we will apply the easy path to similarity measures, feature weighting, and adapation. For example:

- After grouping together rows with similar estimates, we might weight features by their variance within each group (and higher variance means lower weight).
- Alternatively, Lipowezky [?] observes that feature and case selection are similar tasks (both remove cells in the hypercube of all cases times all columns). Under this view, it should be possible to covert our case selector to a feature selector.

Our investigations in this area are very preliminary and, at this time, we have no conclusive results to report.

APPENDIX

With the exception of ISBSG-Banking and SDR, all the data used in this study is available at http://promisedata.org/data or from the authors. As shown in Figure **??**, our data includes:

- Data from the International Software Benchmarking Standards Group (ISBSG);
- The Desharnais and Albrecht data sets;
- SDR, which is data from projects of various software companies from Turkey. SDR is collected from Softlab, the Bogazici University Software Engineering Research Laboratory repository [?];
- And the standard COCOMO data sets (Cocomo*, Nasa*).

Projects in ISBSG dataset can be grouped according to their business domains. In previous studies, breakdown of ISBSG according to business domain has also been used [**?**]. Among different business domains we selected banking due to:

1. Banking domain includes many projects whose data quality is reported to be high (ISBSG contains projects with missing attribute values).
2. ISBSG Banking domain is the dataset we have analyzed and worked for a long time due to our hands on experience in building effort estimation models in banking industry.

We will denote the banking domain subset of ISBSG as "ISBSG-Banking".

Note that two of these data sets (Nasa93c2, Nasa93c5) come from different development centers around the United States. Another two of these data sets (Cocomo81e, Cocomo81o) represent different kinds of projects:

- The Cocomo81e "embedded projects" are those developed within tight constraints (hardware, software, operational, ...);
- The Cocomo81o "organic projects" come from small teams with good experience working with less than rigid requirements.

Note also in Figure **??**, the skewness of our effort values (2.0 to 4.4): our datasets are extremely heterogeneous with as much as 40-fold variation. There is also some divergence in the features used to describe our data:

- While our data includes some effort value (measured in terms of months or hours), no other feature is shared by all data sets.
- The Cocomo* and NASA* data sets all use the features defined by Boehm [?]; e.g. analyst capability, required software reliability, memory constraints, and use of software tools.
- The other data sets use a wide variety of features including, number of entities in the data model, number of basic logical transactions, query count and number of distinct business units serviced.