Transfer Learning with Bellwethers to find Good Configurations

Vivek Nair, Rahul Krishna, Tim Menzies, and Pooyan Jamshidi*

Computer Science, NC State, USA, *Carnegie Mellon University, USA

{vivekaxl,i.m.ralk,tim.menzies,pooyan.jamshidi}@gmail.com

ABSTRACT

As software systems grow in complexity, the space of possible configurations grows exponentially. Within this increasing complexity, developers, maintainers, and users cannot keep track of the interactions between all the various configuration options.

Finding the optimally performing configuration of a software system for a given setting is challenging. Recent approaches address this challenge by learning performance models based on a sample set of configurations. However, collecting enough data on enough sample configurations can be very expensive since each such sample requires configuring, compiling and executing the entire system against a complex test suite.

The central insight of this paper is that choosing a suitable source (a.k.a. "bellwether") to learn from, plus a simple transfer learning scheme will often outperform much more complex transfer learning methods. Using this insight, this paper proposes BEETLE, a novel bellwether based transfer learning scheme, which can identify a suitable source and use it to find near-optimal configurations of a software system. BEETLE significantly reduces the cost (in terms of the number of measurements of sample configuration) to build performance models. We evaluate our approach with 61 scenarios based on 5 software systems and demonstrate that BEETLE is beneficial in all cases.

This approach offers a new highwater mark in configuring software systems. Specifically, BEETLE can find configurations that are as good or better as those found by anything else while requiring only $\frac{1}{7}$th of the evaluations needed by the state-of-the-art.

KEYWORDS

Performance Optimization, SBSE, Transfer Learning, Bellwether

1 INTRODUCTION

Finding the right set of configurations that can achieve the best performance becomes increasingly challenging as the number of configurations increases [57]. As software systems grow in both size and complexity, optimizing a software system to meet the needs of a workload has surpassed the abilities of humans [4].

Much current research has explored this problem, usually by creating accurate performance models that predict performance characteristics. While this approach is cheaper and more effective than manual configuration, it still incurs the expensive of extensive data collection about the software [13, 14, 19, 36–38, 41, 47, 49]. This is undesirable, since this data collection has to be repeated if ever the software is updated on the workload of the system changes abruptly. In such a scenario, all prior research suffers from the same drawback:

> These approaches do not learn from previous optimization experiments and must be rerun whenever the environment of the experiments change.

Note our use of the term *environment*. This refers to the external factors influencing the performance of the system such as workload, hardware, version of the software.

Recent research in performance prediction for configurable systems has shown that *transfer learning* can be effective for resolving 'cold start' problems—problems for which collecting data is expensive. Transfer learning typically entails the transfer of information from a selected "*source*" software system operating in an environment to learn a predictive model for predicting the performance of the system in the "*target*" environment (software system in a different environment), hence enabling a user to reuse measurements. This approach has received much attention in the software analytics literature[22, 24, 25, 27, 39, 40, 44, 51].

Transfer learning can only be useful in cases where the source environment is similar to the target environment. If the source and the target are not similar, knowledge should not be transferred. In such extreme situation, transfer learning can be unsuccessful and can lead to a *negative transfer*. Prior work on transfer learning focused on "*What to transfer*" and "*How to transfer*", by implicitly assuming that the source and target are related to each other. Hence, that work failed to address "*When to transfer*" [42]. Jamshidi et al. [20] alluded to this and explained when transfer learning works but, did not provide a method which can help in selecting a suitable source. In this paper, we focus on solving the problem of performance optimization by choosing a suitable source to transfer knowledge. This has not been explored in the prior work.

The issue of identifying a suitable source is a common problem in transfer learning. To address this, some researchers [27, 28, 31, 32] have recently proposed the use of the *bellwether* effect, which states that:

> " ... When analyzing a community of software projects, then within that community there exists at least one exemplary project, called the bellwether(s), which can best define predictors for other projects ..."

The *bellwether effect* has shown promise in identifying suitable sources for transfer learning in such varied domains as defect prediction, effort estimation, and code smell detection [27]. However, the effect was shown only to work on domains where the data is relatively generic, easy to gather, and free of budget constraints [27]. In this paper, we introduce BEETLE, a bellwether based transfer learner, which uses the bellwether effect to identify suitable sources to train transfer learners for performance optimization. Our main claim of this paper is:

> A *good source* with a *simple transfer learner* is better than source agnostic *complex* transfer learners.

In summary, we make the following contributions:

(1) *Source selection using bellwethers:* We show that the *bellwether effect* exists in performance optimization and that we can use this

to discover suitable sources (called bellwether environments) to perform transfer learning. (§8)

(2) *A fast novel source selection algorithm:* We develop a fast algorithm for discovering the bellwether environment with only ≈ 10% of the measurements (§5).

(3) *Transfer learning using Bellwethers:* We develop a novel transfer learning algorithm using Bellwether called BEETLE (short for Bellwether Transfer Learner) that uses the bellwether environment to construct a simple transfer learner (§5).

(4) *More effective than non-transfer learning:* We show that using the BEETLE is just as good as than non-transfer learning approaches. It is also significantly more economical. (§8)

(5) *More effective than state-of-the-art methods:* We show that the configurations discovered using the bellwether environment are much closer to the true-optima when compared to other state-of-the-art methods [21, 53]. And we show that we are a lot more economical. (§8).

The rest of the paper is structured as follows. In §2, provides motivations for this work. §3 introduces the research questions. In §4, we provide formal definitions of terminologies used in this paper. §5 presents BEETLE, a new transfer learner proposed in this paper. In §6, two state-of-the-art transfer learners are discussed. §7 contains the experimental setup. In §8, the results of the paper are presented as answers to research questions. We discuss further implications of our findings in §9. Threats of validity is highlighted in §10. Finally, we conclude our findings in §11.

2 MOTIVATION

This section motivates our work by highlighting the many problems associated with performance modeling and transfer learning.

Modern software systems come with a large number of configuration options. For example, in APACHE (a popular web server) there are around 600 different configuration options and in HADOOP, as of version 2.0.0, there are around 150 different configuration options [57]. Previous empirical studies have also shown that the number of options is growing over releases [57]. These configuration options control the internal properties of the system such as memory, response times. The number of configuration options usually increase over time [54, 57]. Given the **large number of configurations, it becomes increasingly difficult to assess the impact of the configuration options** of the performance of the software system. To address this issue, a common practice is to employ performance prediction models constructed using machine learning algorithms to estimate the performance of the system under these configurations [13, 15, 18, 50, 52, 56].

Further, research has shown that, when faced with a **large volume of configuration options, developers tend to ignore a majority (over 80%) of the configuration options** [57]. This leaves a considerable amount of untapped potential and often induces poor performance of software systems [57]. To leverage the full benefit of the software system by exploiting the flexibility of the features offered by the system, researchers augment performance prediction models to enable performance optimization [37, 41]. Performance optimization extends performance prediction by identifying the best set of configuration options to pick to accomplish a given task with near-optimal performance.
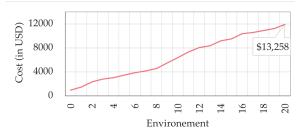

Figure 1: Cost incurred for performance measurements of 2048 configuration options for x264**. Measurements were made on Amazon AWS** c4.large **cluster at $0.0252/hour.**

Performance optimization requires access to measurements of the software system under various configuration settings. However, obtaining these **performance measurements can take a significant amount of time and cost**. For example, in one of the software systems studied here (a video encoding application called x264), it takes over 1536 hours to obtain performance measurements for 11 out the 16 possible configuration options [53]. This is in addition to other time-consuming tasks involved in commissioning these systems such as setup, teardown, etc. Further, making performance measurements can cost an exorbitant amount of money. For the same system, Figure 1 shows the amount we spent on gathering performance measurements on 2048 different configurations.

For a software system under a new environment, instead of having to make exhaustive cost and time intensive measurements, it makes sense to **reuse performance measurements made for previous environments**. The concept of reusing information from other sources is the idea behind *transfer learning* [22, 24, 25, 39, 40, 44, 51]. Specifically, to predict for the optimum configurations in a new environment (referred to as *target environment*), we may use the performance measures of another workload as a proxy (referred to as the *source environment*). For performance optimization, such transfer learning approaches have been shown to decrease the cost of learning by a significant amount [8, 20, 21, 53].

It must be noted that **transfer learning methods place an implicit faith in the nature of the source**. Several researchers in transfer learning caution that the source must be chosen with care to ensure optimum performance of transfer learners [1, 30, 58]. An incorrect choice of source may result in the all too common *negative transfer* phenomenon [1, 3, 43, 46]. A negative transfer can be particularly damaging in that it often leads to performance degradation instead of performance optimization [1, 20]. A preferred way to avoid negative transfer is with *source selection* [1, 27, 28]. In software engineering, researchers have shown that the so-called *bellwether effect* can be used to identify source datasets for effective transfer learning [28]. This bellwether effect has been shown to be very effective in defect prediction, effort estimation, code-smell detection, etc. [27, 31, 32].

In this work, we introduce the notion of source selection with bellwether effect for transfer learning in performance optimization. With this, we develop a Bellwether Transfer Learner called BEETLE. We show that, for performance optimization, BEETLE can outperform both non-transfer learning methods and the current state-of-the-art transfer learning methods.

3 RESEARCH QUESTIONS

This inquiry is structured around the following research questions.

RQ1: Does there exist a Bellwether Environment?

Purpose: In the first research question, we ask if there exist bellwether environments to train transfer learners for performance optimization. We hypothesize that, if these bellwether environments exist, we can improve the efficacy of transfer learning algorithms.

Approach: To answer this research question, we explore five popular open source software systems (for details see §7.1 and Table 1). These software systems have performance measurements under different environments. In each of these software systems, we train a transfer learning model on one environment (the source) and predict for optima in all the other environments. We repeat this process in a round-robin manner for every environment. We then statistically rank the source environments based on their ability to find near-optimal solutions on the targets. If there exist bellwether environments, then those bellwether environments will have a better rank compared to all others environments.

> ***Result:*** We find that bellwether environments are prevalent in performance optimization. That is, in each of the software systems, there exists at least one environment that can be used to construct superior transfer learners.

RQ2: How many performance measurements are required to discover bellwether environments?

Purpose: Having established that bellwether environments are prevalent, the purpose of this research question is to establish how many performance measurements need to be made in the environments to discover bellwether environments.

Approach: To answer this question, we developed an iterative method, based on incremental sampling strategy to find the bellwether environment. We start with 1% of configurations from each environment and incrementally increases the number of sampled until we find the bellwether. Before each increment, we eliminate those environments which do not show much promise.

> ***Result:*** We can discover a potential bellwether environment by measuring as little as 10% of the total configurations across all the software system.

RQ3: How does BEETLE compare to non-transfer learning methods?

Purpose: The alternative to transfer learning is just to use the target data (similar to methods proposed in prior work) to find the near-optimal configurations. Literature is abundant with performance optimization algorithms that do not use transfer learning [13, 36, 37, 47]. For our comparisons, we used the performance optimization model proposed by Nair et al. [36] in FSE '17.

Approach: To answer this research question we compute the Win-Loss ratios of transfer learning with the bellwether environment (aka. BEETLE) to a regular performance optimization method. In addition to this, we compare the cost of the methods, in terms of number of measurements of learning a model.

> ***Result:*** Our experiments demonstrate that transfer learning using bellwethers (BEETLE) outperforms non-transfer learning methods both in terms of cost and the quality of the model.

RQ4: How does BEETLE compare to state-of-the-art methods?

Purpose: In this research question we compare BEETLE with two other state-of-the-art transfer learners used commonly in performance optimization (for details see §6). The purpose of this research question is to determine if a simple transfer learner like BEETLE with carefully selected source environments can perform as well as other complex transfer learning algorithms that do not perform source selection.

Approach: To answer this question, we compare the performance of the near-optimal configurations found using the bellwether environment to the near-optimal configuration found by other transfer learning methods. The configuration found by the bellwether environment is similar to (or better than) the other transfer learning methods.

> ***Result:*** We show that a simple transfer learning using bellwether environment (BEETLE) just as good as (or better than) current state-of-the-art transfer learners.

4 PROBLEM FORMALIZATION

Environments: A software system ($\mathcal{A}$) has n configuration options which can be tweaked to change the performance of the software system (Y). The software system can be operated in different environments ($e \in E$). Each environment is described by 3 variables $\{h, w, v\}$ drawn from the environment space. Here, $h \in H$ represents the hardware, $w \in W$ represents the workload, and $v \in V$ represents the software version. In a software system, the total number of environments E is given by $E = H \times W \times V$. A software system ($\mathcal{A}$) operating in an environment ($e \in E$) is denoted by $\mathcal{A}_e$.

Configuration: Let C_i indicate the ith configuration option of a software system $\mathcal{A}$ operating in environment e (denoted by $\mathcal{A}_e$), which can either be (1) numeric or (2) boolean. A configuration c^i is a member of the configuration space C. C is a Cartesian product of all possible options $C = \text{Dom}(C_1) \times \text{Dom}(C_2) \times ... \times \text{Dom}(C_n)$, where $\text{Dom}(C_i) = \{0, 1\}$ (in our setting) and n is the number of configuration options.
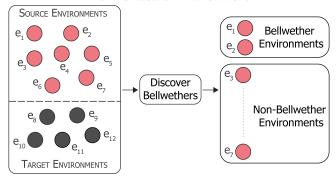
Performance: Each configuration (C) of a system, $\mathcal{A}_e$, has a corresponding performance measure $y \in Y_{A,e}$ associated with it. The configuration and the corresponding performance measure is referred to as independent and dependent variables respectively. We denote the performance measure associated with a given configuration (c^i) by $y = f(c^i)$. We consider the problem of finding the near-optimal configurations (c^*) such that $f(c^*)$ is less than other configurations in $C_{A,h,w,v}$. That is:

$$f(c*) \leq f(c) \; \forall c \in C_{A,h,w,v} \setminus c^* \quad \text{for minimizing objective}$$
$$f(c*) \geq f(c) \; \forall c \in C_{A,h,w,v} \setminus c^* \quad \text{for maximizing objective}$$

Transfer Learning: In transfer learning, we find the near-optimal configuration for a target environment (A_{e_t}), by learning from the measurements ($< c, y >$) for the same system operating in different source environments (A_{e_s}).

Bellwether Environment: We show that, when performing transfer learning, there are exemplar source environments called the bellwether environment(s) ($\mathcal{B} = \{e_{s1}, e_{s2}, ..., e_{sn}\} \subset E$), which are the best source environment(s) to find near-optimal configuration for the rest of the environments ($\forall e \in E \setminus \mathcal{B}$).

Pick Bellwether Environment

Figure 2: This figure demonstrates how to pick the bellwethers

```
1   def FindBellwether(sources, step_size, budget, thres, lives):
2     while lives or cost > budget:
3       "Sample configurations"
4       sampled = list()
5       for source in sources:
6         sampled += source.sample(step_size)
7       "Get cost"
8       cost = get_cost(sampled)
9       "Evaluate pair−wise performances"
10      perf = get_perf(sampled)
11      "Remove non−bellwether environments"
12      sources=remove_non_bellwethers(sources, perf, thres)
13      "Loose life if no sources are removed"
14      if prev == len(sources): lives −= 1
15    "Return a bellwether"
16    return sources[argmin(perf)]
```

Figure 2: This figure demonstrates how to pick the bellwethers

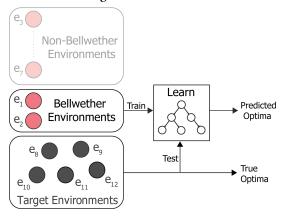## 5   BEETLE: BELLWETHER TRANSFER LEARNER

In this paper, we propose an alternative transfer learning approach to the current state-of-the-art discussed in the previous section. Our approach has two key components:

(1) *Identifying the bellwether environment:* To train a transfer model, we use bellwether effect to discover the best source environments (known as the *bellwether environment*) among the available environments.

(2) *Construct the Transfer Model:* Next, to perform transfer learning, we use these bellwether environments to train a performance prediction model with regression tree [7].

Our key finding is that if a source environment is carefully selected using the bellwether effect, then it is possible to build a simple transfer model without any complex methods and still be able to generate near-optimal configurations in a target environment.

In Figures 2 & 3, we describe BEETLE and list a generic algorithm of BEETLE. In this example, there are 7 source environments ($e_1, e_2, ..., e_7$), which have been optimized previously. $e_8, e_9, ..., e_{12}$ represents the target environments, which need to be optimized. BEETLE'objective is to find a bellwether among the source environments and use it to find the near-optimal configuration for the target environments. BEETLE, a Bellwether based approach can be separated into the following main steps: (i) finding the Bellwether environment, and (ii) using the Bellwether environment to find the near-optimal configuration for target environments.

Transfer Learning with Bellwether Environment

Figure 3: This figure demonstrates how to construct BEETLE.

```
1   def BEETLE(sources, target, budget):
2     "Find the bellwether environment"
3     bellwether = FindBellwether(sources, step_size, budget, ↪
        thres, lives)
4     "Sample the bellwether source to fit budget"
5     b_some = bellwether.sample(budget)
6     "Train a prediction model with the bellwether"
7     prediction_model = regTree.train(b_some)
8     predicted = prediction_model.test(target.indep)
9     return target[argmin(predicted)]
```

Figure 3: This figure demonstrates how to construct BEETLE.

5.1   Finding Bellwether Environments

The central idea for finding Bellwether is to use minimal sampling to recursively eliminate environments which do not show promise of being a bellwether, that is those environments cannot be used to predict the near-optimal solutions. Figure 2 is a generic algorithm that defines the process of finding bellwethers. The process starts by sampling a small subset of the source environments. The size of the subset is controlled by a predefined parameter *step_size* (Line 6). The cost of sampling the configuration is calculated (Line 8). In our setting, we use the number of measurements as a proxy for cost and can be replaced by any user-defined cost function (*get_cost*). To compute the effectiveness of an environment, the sampling configurations along with the performance measure is used to build a performance model (regression tree in our setting). Please note that we choose regression tree as a model because it has been extensively used for performance prediction of configurable software systems and demonstrated good results [13, 14, 36–38, 47, 53]. This model is then used to predict the optimal configuration among the configurations sampled in Line 6 (Line 10). We only used the sampled configurations because the actual performance of a source can only be calculated if the actual performance values (associated with the configurations) are known. This process is repeated for all the environments (represented as *sources*) under consideration. Depending on how an environment can find the near-optimal configuration for other environments and a user-defined threshold (*thres*), the non-bellwether environments are eliminated (Line 12). Non-bellwether environments are environments, which are not able to find near-optimal configurations for the other environments. If the no environment is eliminated (with more data) when compared

to the previous iteration (lesser data), then a life is lost (Line 14). When all lives are expired or run out of the budget, the search process terminates. The environment with maximum performance is returned as the bellwether (Line 16). Please note that, *FindBellwether* can identify multiple bellwethers (e_1, e_2). However, in our setting, we return a single bellwether.

The motivation behind using the parameter lives is to detect convergence of the search process. If adding more configuration does not improve the chances of finding the Bellwether, the search process should terminate to avoid resource wastage; see also § 9.

5.2 Using Bellwethers

Once the bellwether is identified, it can be used to find the near-optimal configurations of target environments. As shown in Figure 3 *FindBellwether* returns the predicted bellwether environment (Line 3). Performance modeling then samples the bellwether environments (Line 5) for some number of samples (a user-defined parameter called budget). Please note that a user might choose to reuse the measurement used in *FindBellwether* and save on cost. The sampled configuration and their corresponding performance measures it used to build a prediction model (Line 7). Similar to *FindBellwether*, we use regression tree as our modeling method of choice. The prediction model is there used to predict the optimal configuration among the (unevaluated configurations) for the target environment (Lines 8-9). The predicted optimal configuration is returned as the best configuration. This process is then repeated for each target environments ($e_8, e_9, ..., e_{12}$).

6 TRANSFER LEARNING IN PERFORMANCE OPTIMIZATION

In this paper, BEETLE is compared against (a) two state-of-the-art transfer learners from ICPC'17 [54] and SEAMS'17 [21]; and (b) a non-transfer learner from FSE'17 [37].

6.1 Transfer Learning with Linear Regression

Valov et al. [53] proposed an approach for transferring performance models of software systems across platforms with *different hardware settings*. Figure 4 shows the pseudocode of the transfer learning method. The method consists of the following two components:
- *Performance prediction model:* The configuration source hardware are sampled using *Sobol* sampling. The number of configurations is given by $T \times N_f$, where $T = \{3, 4, 5\}$ is the *training coefficient* and N_f is the number of configuration options. These configurations are used to construct a *Regression Tree* model.
- *Transfer Model:* To transfer the predictions from the source to the target, the authors construct a linear regression model since it was found to provide good approximations of the transfer function. To construct this model, a small number of random configurations are obtained from the source and the target.

6.2 Transfer Learning with Gaussian Process

Jamshidi et al. [21] took a slightly different approach to transfer learning. They used Gaussian Processes (GP) to find the relatedness between the performance measures in source and the target as well as the configurations. The relationships between input configurations were captured in the GP model using a covariance matrix that defined the kernel function to construct the Gaussian processes

```
def LinearTransform(source, target,                    1
                    training_coef, budget):            2
    "Construct a prediction model"                     3
    prediction_model=regTree.train(source, training_coef)  4
    "Sample random measurements"                       5
    s_samp = source.sample(budget)                     6
    t_samp = target.sample(budget)                     7
    "Get performance measurements"                     8
    s_perf=get_perf(s_samp)                            9
    t_perf=get_perf(t_samp)                            10
    "Train a transfer model with LR"                   11
    transfer_model=linear_model.train(s_perf, t_perf)  12
    return prediction_model, transfer_model            13
```

Figure 4: Linear Transformation Transfer. From Valov et al. [53].

```
def GPTransform(source, target,                        1
                source_budget, target_budget):         2
    "Sample random configurations"                     3
    s_some = source.sample(source_budget)              4
    t_some = target.sample(target_budget)              5
    "Get performance measurements"                     6
    s_perf = get_perf(s_some)                          7
    t_perf=get_perf(t_some)                            8
    "Compute correlation"                              9
    perf_correlation = get_correlation(s_perf, t_perf) 10
    "Compute covariance"                               11
    input_covariance = get_covariance(s_some, t_some)  12
    "Construct a kernel"                               13
    kernel = input_covariance × perf_correlation       14
    "Train the Gaussian Process model"                 15
    learner = GaussianProcessRegressor(kernel)         16
    prediction_model = learner.train(s_some)           17
    return prediction_model                            18
```

Figure 5: Gaussian Process Transformation Transfer. From Jamshidi et al. [21].

model. To encode the relationships between the measured performance of the configuration in the source the target, the authors propose a scaling factor to the above kernel.

The new kernel function is a defined as follows:

$$k(s, t, f(s), f(t)) = k_t(s, t) \times k_{xx}(f(s), f(t)) \qquad (1)$$

where $k_t(s, t)$ represents the multiplicative scaling factor. $k_t(s, t)$ is given by the correlation between source f(s) and target f(t) function, while k_{xx} is the covariance function for input environments (s & t). The essence of this method is that the kernel captures the interdependence between the source and target configurations and their corresponding performance measurement values.

6.3 Non-Transfer Performance Optimization

A performance optimization model with no transfer was proposed by Nair et al. [37] in FSE '17. It works as follows:
(1) Sample a small set of measurements of configurations from the target environment
(2) Construct performance prediction model with regression trees.
(3) Predict for near-optimal configurations.

The key distinction here is that unlike transfer learners, that use a *different source environment* to build to predict for near-optimal configurations in a target environment, a non-transfer method such as this uses configurations *from within the target* environment to predict for near-optimal configurations.

Table 1: Overview of the real-world subject systems. |C|=Number of Configurations, |c|=Number of configuration options, |E|: Number of Environments, |H|: Hardware, |W|: Workloads, and |V|: Versions. See http://tiny.cc/bw_software **for more details.**

Domain	System	Description	|C|	|c|	|E|	|H|	|W|	|V|
Video Encoder	X264	x264is a video encoder that compresses video files to adjust output quality, encoder types,and encoding heuristics.	4000	16	21	4	3	3
SAT Solver	SPEAR	An industrial strength bit-vector arithmetic decisionprocedure and a Boolean satisfiability (SAT) solver. It is designed for proving software verification conditions and it is used for bug hunting.	16384	14	10	2	4	2
Database	SQLite	x264is a video encoder that compresses video files to adjust output quality, encoder types,and encoding heuristics.	1000	14	15	2	13	2
Compiler	SaC	SQLite is a lightweight relational database management sys-tem, em-bedded in several browsers and operating systems.	846	50	7	2	10	2
Data Analytics	Apache Storm	Apache Storm is a distributed stream processing computation frame-work written predominantly in the Clojure programming language.	2048	12	4	2	3	2

7 EXPERIMENTAL SETUP

7.1 Subject Systems

In this study, we selected five configurable software systems from different domains, with different functionalities, and written in different programming. We selected these real-world software systems since their characteristics cover a broad spectrum of scenarios. Table 1 lists the details of the software systems used here. The rest of this section provides a summary of the subject systems.

SPEAR is an industrial strength bit-vector arithmetic decision procedure and Boolean satisfiability (SAT) solver. It is designed for proving software verification conditions, and it is used for bug hunting. We considered a configuration space with 14 options with 2^{14} or 16384 configurations. We measured how long it takes to solve an SAT problem in all 16,384 configurations in 10 environments.

x264 is a video encoder that compresses video files and has 16 configurations options to adjust output quality, encoder types, and encoding heuristics. Due to the size of the configuration space, we randomly sample 4000 configurations in 21 environments.

SQLite is a lightweight relational database management system, embedded in several browsers and operating systems, which has 14 configuration options to change indexing and features for size compression. Due to a limited budget, we use 1000 randomly selected configurations in 15 different environments.

SaC is a compiler for high-performance computing. The SaC compiler implements a large number of high-level and low-level optimizations to tune programs for efficient parallel executions. It has 50 configuration options to control optimization options. We measure the execution time of a program compiled in 71,267 randomly selected configurations.

STORM is a distributed stream processing framework which is used for data analytics. We run three benchmarks and measure the latency of the benchmark in 2,048 randomly selected configurations to assess the performance impact of Storm's options.

7.2 Evaluation Criterion

Typically, performance models are evaluated based on the accuracy or error using measures such as MMRE. We note that there has been a lot of criticism regarding MMRE, which shows that MMRE along with other accuracy statistics such as MBRE has been shown to cause conclusion instability [10, 34, 35], given by:

$$MMRE = \frac{|predicted - actual|}{actual} \cdot 100$$

While this typical in performance prediction, our objective is to find the near-optimal configurations or *performance optimization*. For this, measures similar to MMRE is not applicable [37]. Recently work by Nair et al. [37]has shown that when MMRE cannot be used, other measures such as rank difference may be used—which emphasizes the sorted order of configurations and their performances rather than accuracy of the predictions.

Once the performance model is trained, the accuracy of the model is measured by sorting the values of $y = f(x)$ from 'small' to 'large', that is:

$$f(c_1) \leq f(c_2) \leq f(c_3) \leq ... \leq f(c_n). \tag{2}$$

The predicted rank order is then compared to the actual rank order. We note that rank difference though effective is not particularly informative since it is sensitive to the workload. This means that in some workloads, a small difference in performance measure can lead to a large rank difference and vice-versa. For example, in SPEAR_0 the top performance of the optimal configuration (rank 1) and 100th best configuration has a difference of 0.09%—which means a large rank difference of 100 does not mean poor performance.

Hence, we define a performance measure called *Normalized Absolute Residual* (NAR), which represents the difference between the actual performance measurements of the optimal configuration and the predicted optimal configuration. The difference between the actual and predicted optimal configuration is normalized to the difference between the actual best and worst configurations,

$$NAR = \frac{min(f(c)) - f(c^*)}{max(f(c)) - min(f(c))} \cdot 100 \tag{3}$$

where $c \in C_{A, h, w, v} \setminus c^*$. This measure is similar to Absolute Residual (lower is better). However, in our setting the range of the performance measures across different environments are not equal (hence the need for a normalization step).

X264

Rank	Dataset	Median	IQR	
1	x264_18	0.35	1.82	
1	x264_9	0.35	1.62	
2	x264_10	0.94	8.25	
2	x264_7	0.94	8.25	
2	x264_11	1.62	7.46	
3	x264_16	2.33	12.18	
3	x264_2	2.33	12.18	
3	x264_6	2.82	5.35	
3	x264_20	3.65	13.74	
4	x264_19	6.95	41.97	
4	x264_3	8.68	49.78	
4	x264_17	13.61	32.32	
4	x264_13	16.42	51.65	
4	x264_15	20.14	50.68	
5	x264_14	27.24	42.74	
5	x264_0	28.63	49.77	

Sac

Rank	Dataset	Median	IQR	
1	sac_6	0.27	0.14	
2	sac_4	0.96	4.26	
2	sac_8	1.04	3.67	
2	sac_9	2.29	4.98	
3	sac_5	10.8	89.65	

Storm

Rank	Dataset	Median	IQR	
1	storm_feature9	0.0	0.0	
1	storm_feature8	0.0	0.0	
1	storm_feature6	0.0	0.01	
1	storm_feature7	0.01	0.04	

Spear

Rank	Dataset	Median	IQR	
1	spear_7	0.1	0.1	
1	spear_6	0.1	0.2	
1	spear_1	0.1	0.1	
1	spear_9	0.1	0.5	
1	spear_8	0.1	0.2	
1	spear_0	0.1	0.91	
2	spear_5	0.28	0.3	
3	spear_4	0.6	1.17	
4	spear_2	1.09	5.31	
5	spear_3	1.89	4.48	

Sqlite

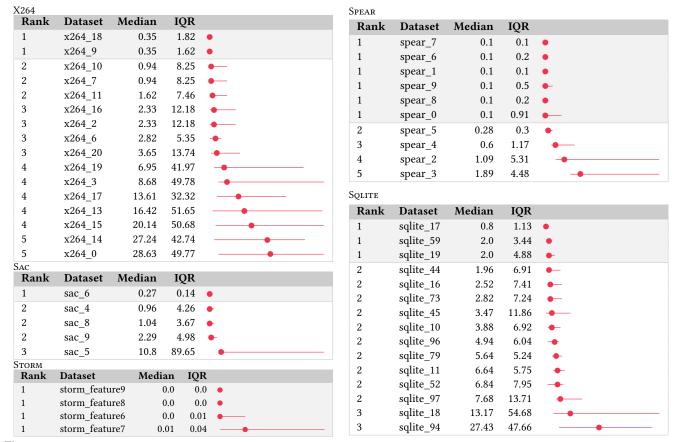| Rank	Dataset	Median	IQR	
1	sqlite_17	0.8	1.13	
1	sqlite_59	2.0	3.44	
1	sqlite_19	2.0	4.88	
2	sqlite_44	1.96	6.91	
2	sqlite_16	2.52	7.41	
2	sqlite_73	2.82	7.24	
2	sqlite_45	3.47	11.86	
2	sqlite_10	3.88	6.92	
2	sqlite_96	4.94	6.04	
2	sqlite_79	5.64	5.24	
2	sqlite_11	6.64	5.75	
2	sqlite_52	6.84	7.95	
2	sqlite_97	7.68	13.71	
3	sqlite_18	13.17	54.68	
3	sqlite_94	27.43	47.66	

Figure 6: Median NAR of 30 repeats. Median NAR is the normalized absolute residual values as described in Equation 7.2, and IQR the difference between 75th percentile and 25th percentile found during multiple repeats. Lines with a dot in the middle (—●—), show the median as a round dot within the IQR. All the results are sorted by the median NAR: a lower median value is better. The left-hand column (*Rank*) ranks the various techniques where lower ranks are better.

7.3 Statistical Validation

Our experiments discussed in RQ1 and RQ4 are all subject to inherent randomness introduced by sampling configurations or by different source and target environments. To overcome this, we use 30 repeated runs, each time with a different random number seed. The repeated runs provide us with a sufficiently large sample size for statistical comparisons. Each repeated run collects the values of NAR to assess the the transfer learners.

To rank these 30 numbers collected as above, we use the Scott-Knott test recommended by Mittas and Angelis [33]. The Skott-Knott test has been endorsed by several SE researchers [2, 23, 26, 29, 45, 48]. Scott-Knott is a non-parametric statistical test that performs a bootstrap test with 95% confidence [9] to determine the existence of statistically significant differences. This followed by an A12 test to check that any observed differences were not trivially small effects [55]. We say that a "small" effect has $a < 0.6$. Scott-Knott test results in treatments being ranked from best to worst. Note that, if a set of treatments are not significantly different, they will have the same ranks.

8 RESULTS

RQ1: Does there exist a Bellwether Environment?

Purpose: The first research question seeks to establish the presence of bellwether environments within different environments of a software system. We hypothesize that, if these bellwether environments exist, we can improve the transfer learning algorithms.

Approach: For each subject software system, we use the environments to perform a pair-wise comparison method (similar to leave one out testing) as follows:

(1) We pick one environment as a source and construct a transfer learner.
(2) Next, we use the remaining environments as targets. For every target environment, we use the transfer learner-constructed in the previous step to predict for the optimum configuration.
(3) Then, we measure the NAR of the predictions (see §7.2).
(4) Afterward, we repeat steps 1, 2, and 3 for all the source environments and gather the outcomes.
(5) Finally, we use Scott-Knott test to rank each environment (and its usefulness as a source).

Vivek Nair, Rahul Krishna, Tim Menzies, and Pooyan Jamshidi*

Table 2: Effectiveness of source selection method.

Subject System	Bellwether Environment		Predicted Bellwether Environment	
	Median	IQR	Median	IQR
SQLite	0.8	1.13	1.8	2.48
Spear	0.1	0.1	0.1	0
X264	0.35	1.62	0.9	1.06
Storm	0.0	0.0	0.0	0.0
Sac	0.27	0.14	0.63	7.4

Summary: Our results are shown in Figure 6. Overall, we find that there is always at least one environment (the bellwether environment) in all the subject systems, that is much superior to others. Note that, Storm is an interesting case, here all the environments are ranked 1, which means that all the environments are equally useful as a bellwether environment. Further, we note that the variance in the bellwether environments are much lower compared to other environments.

> **Result:** There exist environments in each subject system, which act as bellwether environment and hence can be used to find the near-optimal configuration for the rest of the environments.

RQ2: How many performance measurements are required to discover bellwether environments?

Purpose: The bellwether environments found in RQ1 required us to use 100% measurements from all the environments. This may not be economical in a real-world scenario. It can be prohibitively expensive to run and test all configurations of subject systems (as done in RQ1) since their configuration spaces are large. Thus, in this research question, we ask if we can find the bellwether environments sooner using fewer configurations.

Approach: We developed an iterative method, based on incremental sampling strategy to find the bellwether environment. It works as follows

(1) We start from 1% of configurations from each environment and assume that every environment is a potential bellwether environment.

(2) Then, we increment the number of configurations in steps of 1% and measure the NAR values.

(3) We rank the environments and eliminate those that do not show much promise. A detailed description of how this is accomplished can be found in §5.

(4) We repeat the above steps until we cannot eliminate any more environments.

The above strategy uses $\mu + \sigma$ of the NAR values at each step as a threshold to eliminate non-bellwether environments. To function correctly, this requires the NAR values to follow a normal distribution. If normality is violated, we used power transforms to make the data more normal distribution-like. We note that this is a prevalent strategy commonly used in other domains to reduce the size of options or alternatives [6] and is also known as *backward elimination* [5].

To see if our proposed method is effective, we compare the performance of bellwether environment with the predicted bellwether environment.

Summary: Table 2 summarizes our findings. We find that,

- In all 5 cases, using at most 10% of the configurations we find one of the bellwether environments that are found with 100% of the configurations. See, column Rank in Table 2.
- In terms of quality of predictions, the NAR values of the predicted bellwether environments with 10% of the configurations is less than 1.0% different from the bellwether found at 100%.

Our results are encouraging in that they demonstrate how the bellwether environments can be discovered very fast with just a fraction of the original configuration size. Since fewer configuration takes less time to collect and is cheaper, we can assert that discovering bellwether environments can be very economical.

> **Result:** The bellwether environment can be recognized using only a fraction of the measurements (under 10%), and the identified bellwether environments have similar NAR values to the actual bellwether environment.

RQ3: How does BEETLE compare to non-transfer learning methods?

Motivation: Having established that there exist bellwether environments in the subjects systems (RQ1) and that they can be found with very few measurements (RQ2), in this research question we explore how BEETLE compares to a non-transfer learning approach. For our experiment, we use the non-transfer performance optimizer proposed by Nair et al. [37]. More details on Nair et al.'s method can be found in §6.3.

Both BEETLE and Nair et al.'s methods seek to achieve the same goal—find optimal configuration in a target environment. BEETLE uses configurations from a *different source* to achieve this, whereas the non-transfer learner uses configurations from *with the target*.

Approach: Our setup involves evaluating the Win/Loss ratio of BEETLE to the non-transfer learning algorithm while predicting for the optimal configuration. Comparing against true optima, we define "win" as cases where BEETLE has a better (or same) optima as the non-transfer learner. A "loss" otherwise.

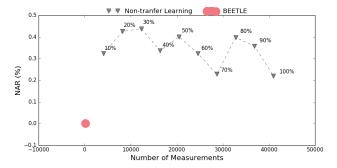**Summary:** Our results are shown in Figures 7 and 8. In Figure 8,

Figure 7: Trade-off of between the quality of the configurations and the cost to build the model in case of X264. We notice that the cost to find a good configuration using bellwethers is much lower than that of Non-transfer learning methods.
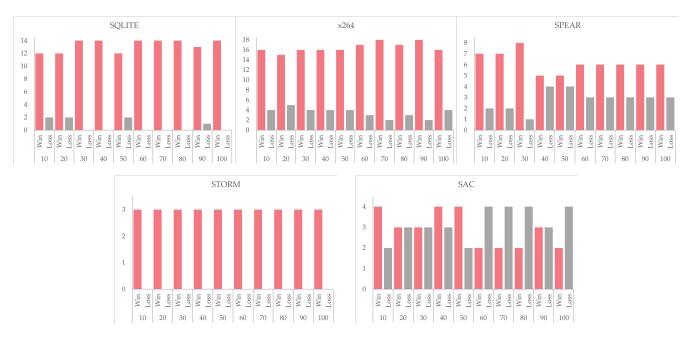
Figure 8: Win/Loss analysis of learning from the bellwether environment and target environment using Scott Knott. The x-axis represents the percentage of data used to build a model and y-axis is the count. BEETLE wins in all models except for SAC– and there only there when we have measured 50+% of the data.

the x-axis represents the number of configurations (expressed in %) to train the non-transfer learner and BEETLE, and the y-axis represents the number of wins/losses. From this figure we observe:

- *Better performance:* In 4 out of 5 systems, the BEETLE "wins" significantly more than it "losses". This means that BEETLE is better than (or at least as good as) non-transfer learning methods.
- *Lower cost:* In terms of cost, we note that BEETLE outperforms the non-transfer learner significantly, "winning" at configurations of 10% to 100% of the original sample size. Further, when we look at the trade-off between performance and number of measurements in Figure 7, we note that BEETLE achieves an NAR close to zero with close around 100 samples. On the other hand, the non-transfer learning method of Nair et al. [37]has significantly larger NAR while also requiring large sample sizes.

> **Result:** BEETLE performs better than (or same as) a non-transfer learning approach. BEETLE is also cost/time efficient as it requires far fewer measurements.

RQ4: How does BEETLE compare to state-of-the-art methods?

Purpose: The main motivation of this work is to show that the source environment can have a significant impact on transfer learning. In this research question, we seek to compare BEETLE with two other state-of-the-art transfer learners by Jamshedi et al. [21] and Valov et al. [53]. For further details of these methods, see §6.

Approach: To perform our comparisons, we use a Scott-Knott test to rank the *NAR* values. These *NAR* values indicate the % performance difference between estimated and the actual near-optimal.

Sac

Rank	Learner	Median	IQR	
1	Jamshidi et al. [21]	1.58	5.39	
2	Baseline	6.89	99.1	
2	Valov et al. [53]	6.99	99.24	

Spear

Rank	Learner	Median	IQR	
1	Jamshidi et al. [21]	0.70	1.29	
1	BEETLE	0.79	1.40	
1	Valov et al. [53]	1.11	1.98	

SQLite

Rank	Learner	Median	IQR	
1	BEETLE	5.41	9.28	
2	Valov et al. [53]	6.96	12.91	
3	Jamshidi et al. [21]	18.51	50.85	

Storm

Rank	Learner	Median	IQR	
1	BEETLE	0.04	0.06	
1	Jamshidi et al. [21]	0.86	20.69	
2	Valov et al. [53]	2.47	53.98	

x264

Rank	Learner	Median	IQR	
1	BEETLE	8.67	27.01	
2	Valov et al. [53]	16.99	41.24	
3	Jamshidi et al. [21]	43.58	28.39	

Figure 9: Comparison between state-of-the-art transfer learners and BEETLE. The best transfer learner is shaded gray .

Summary: Our results are shown in Figure 9. In this figure, the best transfer learner is ranked 1. We note that in 4 out of 5 cases, the baseline transfer learner based on source selection performs just as well as (or better than) the state-of-the-art. This result is encouraging in that it points to significant impact choosing a good
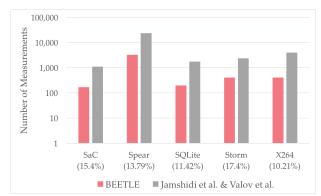
Figure 10: BEETLE v/s state-of-the-art transfer learners. The numbers in parenthesis represent the numbers of measurements BEETLE uses in comparison to the state-of-the-art learners.
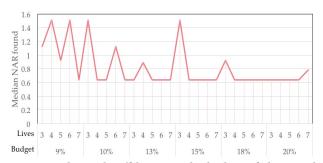

Figure 11: The trade-off between the budget of the search, the number of lives, and the NAR (quality) of the solutions.

source environment can have on the performance of transfer learners. Further, in Figure 10 we compare the number of performance measurements required to construct the transfer learners (note the logarithmic scale on the vertical axis). It can be noticed that BEETLE requires far fewer measurements compared to the other transfer-learning methods.

> **Result:** In most software systems, BEETLE performs just as well as (or better than) other state-of-the-art transfer learners for performance optimization using far fewer measurements.

9 DISCUSSION

What is the trade-off between hyper-parameters and effectiveness of BEETLE? In Figure 11, we show the trade-off between the hyperparameters (budget, lives) and NAR values (effectiveness). We note that the performance is correlated to the budget and number of lives. That is, as budget increases the NAR value decreases. Since our objective is to minimize the number of measurements while reducing overall NAR, we assign the value of 5 to lives and 10% to budget for our experiments.

When are bellwethers ineffective? Existence or discovery of bellwethers depends on the following: (a) *Metrics used:* Finding bellwether using metrics that are not justifiable, may be unsuccessful, e.g, trying to discover bellwethers in performance optimization, by measuring MMRE instead of NAR for will fail (see

http://tiny.cc/bw_metrics) [37]; (b) *Different Software System:* Bellwethers of a certain software system 'A' may not work for software system 'B'; and (c) *Different Performance Measures:* Bellwether discovered for one performance measure (time) may not work for other performance measures (throughput).

Is BEETLE applicable in other domains? Yes, BEETLE can be applied to any transfer learning application, where the choice of the source data impacts the performance of transfer learning. This can be applied to problems such as configuring big data systems [19], finding suitable cloud configuration for a workload [16, 17], configuring hyper parameters of machine learning algorithms [1, 11, 12].

10 THREATS TO VALIDITY

External validity: We selected a diverse set of subject systems and a large number of selected environment changes, but, as usual, one has to be careful when generalizing to other subject systems and environment changes. Even though we tried to run our experiment on a variety of software systems from different domains, we cannot generalize our results beyond these software systems.

Internal validity: Due to the size of configuration spaces, we could only measure configurations exhaustively in one subject system and had to rely on sampling (with substantial sampling size) for the others, which may miss effects in parts of the configuration space that we did not sample. We did not encounter any surprisingly different observation in our exhaustively measured SPEAR dataset. Measurement noise in benchmarks can be reduced but not avoided. We performed benchmarks on dedicated systems and repeated each measurement 3 times. We repeated experiments when we encountered unusually large deviations.

Parameter bias: With all the transfer learners and predictors discussed here, there are a number of internal parameters that have been set by default. The result of changing these parameters may (or may not) have a significant impact on the outcomes of this study.

11 CONCLUSION

Our approach exploits the bellwether effect—there are one or more bellwether environments which can be used to find good configurations for rest of the environments. We also propose a new transfer learning method, called BEETLE, which exploits this phenomenon. We show that BEETLE can quickly identify the bellwether environments with only a few measurements ($\approx 10\%$) and use it to find the near-optimal solutions in the target environments. We have done extensive experiments with 5 highly configurable systems demonstrating that BEETLE can (i) identify the most suitable source to construct transfer learners, (ii) find near-optimal configurations with only a small number of measurements (less than $13.5\% \approx 1/7^{th}$ of configuration space), (iii) performs as well as non-transfer learning approaches , and (iv) performs as well as state-of-the-art transfer learners.

REFERENCES

[1] Muhammad Jamal Afridi, Arun Ross, and Erik M Shapiro. 2018. On automated source selection for transfer learning in convolutional neural networks. *Journal of Pattern Recognition* (2018).

[2] A. Arcuri and L. Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ICSE'11*. 1–10.

[3] Shai Ben-David and Reba Schuller. 2003. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*. Springer.

[4] Phil Bernstein, Michael Brodie, Stefano Ceri, David DeWitt, Mike Franklin, Hector Garcia-Molina, Jim Gray, Jerry Held, Joe Hellerstein, HV Jagadish, and others. 1998. The Asilomar report on database research. *ACM Sigmod record* (1998).

[5] Avrim L Blum and Pat Langley. 1997. Selection of relevant features and examples in machine learning. *Artificial intelligence* (1997).

[6] Christian Borgelt. 2005. Keeping things simple: finding frequent item sets by recursive elimination. In *International workshop on open source data mining: frequent pattern mining implementations*. ACM.

[7] Leo Breiman. 1996. Bagging predictors. *Machine learning* (1996).

[8] Haifeng Chen, Wenxuan Zhang, and Guofei Jiang. 2011. Experience transfer for the configuration tuning in large-scale computing systems. *Transactions on Knowledge and Data Engineering* (2011).

[9] Bradley Efron and Robert J Tibshirani. 1993. *An introduction to the bootstrap*.

[10] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. 2003. A Simulation Study of the Model Evaluation Criterion MMRE. *IEEE Transactions on Software Engineering* (2003).

[11] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* (2016).

[12] Wei Fu, Vivek Nair, and Tim Menzies. 2016. Why is Differential Evolution Better than Grid Search for Tuning Defect Predictors? *arXiv:1609.02613* (2016).

[13] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *International Conference on Automated Software Engineering*. IEEE.

[14] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2017. Data-efficient performance learning for configurable systems. *Empirical Software Engineering* (2017).

[15] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. 2006. Performance prediction based on inherent program similarity. In *International Conference on Parallel Architectures and Compilation Techniques*. IEEE.

[16] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. 2017. Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM. *arXiv preprint arXiv:1712.10081* (2017).

[17] C.-J. Hsu, V. Nair, T. Menzies, and V. W. Freeh. 2018. Scout: An Experienced Guide to Find the Best Cloud Configuration. *arXiv preprint arXiv:1803.01296* (2018).

[18] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* (2014).

[19] Pooyan Jamshidi and Giuliano Casale. 2016. An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems. In *Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*.

[20] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *International Conference on Automated Software Engineering*. IEEE Press.

[21] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer learning for improving model predictions in highly configurable software. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE.

[22] Xiaoyuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. 2015. Heterogeneous Cross-Company Defect Prediction by Unified Metric Representation and CCA-Based Transfer Learning Categories and Subject Descriptors. *Foundations of Software Engineering* (2015).

[23] Vigdis By Kampenes, Tore Dybå, Jo Erskine Hannay, and Dag I. K. Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Information & Software Technology* (2007).

[24] Ekrem Kocaguneli and Tim Menzies. 2011. How to find relevant data for effort estimation?. In *Empirical Software Engineering and Measurement*. IEEE.

[25] Ekrem Kocaguneli, Tim Menzies, and Emilia Mendes. 2015. Transfer learning in effort estimation. *Empirical Software Engineering* (2015).

[26] Ekrem Kocaguneli, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, and Tim Menzies. Distributed development considered harmful?. In *International Conference on Software Engineering*.

[27] Rahul Krishna and Tim Menzies. 2017. Bellwethers: A Baseline Method For Transfer Learning. *arXiv:1703.06218* (2017).

[28] Rahul Krishna, Tim Menzies, and Wei Fu. 2016. Too Much Automation? The Bellwether Effect and Its Implications for Transfer Learning. In *International Conference on Automated Software Engineering*. ACM.

[29] Nancy L Leech and Anthony J Onwuegbuzie. 2002. A Call for Greater Use of Nonparametric Statistics.. In *Annual Meeting of the Mid-South Educational Research Association*. ERIC.

[30] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. 2015. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*.

[31] Solomon Mensah, Jacky Keung, Michael Franklin Bosu, Kwabena Ebo Bennin, and Patrick Kwaku Kudjo. 2017. A Stratification and Sampling Model for Bellwether Moving Window. In *International Conference on Software Engineering and Knowledge Engineering*.

[32] Solomon Mensah, Jacky Keung, Stephen G. MacDonell, Michael Franklin Bosu, and Kwabena Ebo Bennin. 2017. Investigating the Significance of Bellwether Effect to Improve Software Effort Estimation. In *International Conference on Software Quality, Reliability and Security, QRS*. IEEE.

[33] Nikolaos Mittas and Lefteris Angelis. 2013. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *Transactions on Software Engineering* (2013).

[34] I. Myrtveit and E. Stensrud. 2012. Validity and Reliability of Evaluation Procedures in Comparative Studies of Effort Prediction Models. *Empirical Software Engineering* (2012).

[35] I. Myrtveit, E. Stensrud, and M. Shepperd. 2005. Reliability and Validity in Comparative Studies of Software Prediction Models. *IEEE Transactions on Software Engineering* (2005).

[36] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Faster discovery of faster system configurations with spectral learning. *Automated Software Engineering* (2017).

[37] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using bad learners to find good configurations. *Foundations of Software Engineering* (2017).

[38] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding Faster Configurations using FLASH. *arXiv preprint arXiv:1801.02175* (2018).

[39] Jaechang Nam and Sunghun Kim. 2015. Heterogeneous defect prediction. In *Foundations of Software Engineering*. ACM Press.

[40] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *International Conference on Software Engineering*. IEEE.

[41] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Foundations of Software Engineering*. ACM.

[42] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *Transactions on knowledge and data engineering* (2010).

[43] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *Transactions on knowledge and data engineering* (2010).

[44] Fayola Peters, Tim Menzies, and Lucas Layman. 2015. LACE2: Better privacy-preserving data sharing for cross project defect prediction. In *International Conference on Software Engineering*.

[45] Simon Poulding and John A Clark. 2010. Efficient software verification: Statistical testing using automated search. *Transactions on Software Engineering* (2010).

[46] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. 2005. To transfer or not to transfer. In *NIPS 2005 workshop on TL*.

[47] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-efficient sampling for performance prediction of configurable systems (t). In *International Conference on Automated Software Engineering*. IEEE.

[48] Martin J. Shepperd and Steven G. MacDonell. 2012. Evaluating prediction systems in software project estimation. *Information & Software Technology* (2012).

[49] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *International Conference on Software Engineering*. IEEE.

[50] Eno Thereska, Bjoern Doebel, Alice X Zheng, and Peter Nobel. 2010. Practical performance models for complex, popular applications. In *ACM SIGMETRICS Performance Evaluation Review*. ACM.

[51] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* (2009).

[52] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. 2015. Empirical comparison of regression methods for variability-aware performance prediction. In *International Conference on Software Product Line*. ACM.

[53] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring performance prediction models across different hardware platforms. In *International Conference on Performance Engineering*. ACM.

[54] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *International Conference on Management of Data*. ACM.

[55] A. Vargha and H. D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* (2000).

[56] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. 2012. Automated inference of goal-oriented performance prediction functions. In *International Conference on Automated Software Engineering*. ACM.

[57] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Foundations of Software Engineering*. ACM.

[58] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*.