Enabling Portable Building Applications through Automated Metadata Transformation

Arka Bhattacharya
David E. Culler
Jorge Ortiz
Dezhi Hong
Kamin Whitehouse

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-159
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-159.html

August 19, 2014

Enabling Portable Building Applications through Automated Metadata Transformation

Arka Bhattacharya [*], David Culler
Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley
arka,culler@eecs.berkeley.edu

Dezhi Hong, Kamin Whitehouse
Dept. of Computer Science
University of Virginia
hong,whitehouse@virginia.edu

Jorge Ortiz
IBM TJ Watson Research Center
jjortiz@us.ibm.com

ABSTRACT

Sensor network research has facilitated advancements in various domains, such as industrial monitoring, environmental sensing, etc., and research challenges have shifted from creating infrastructure to utilizing it. Commercial buildings provide a valuable setting for investigating automated metadata acquisition and augmentation, as they typically comprise large sensor networks, but have limited, obscure 'tags' that are often meaningful only to the facility managers. Moreover, this primitive metadata is imprecise and varies across vendors and deployments. Extracting meaningful information from a building's sensor data, or control applications using the data, depends on the metadata available to interpret it, whether provided by novel networks or legacy instrumentation.

This state-of-the-art is a fundamental barrier to scaling analytics or intelligent control across the building stock, as even the basic steps involve labor intensive manual efforts by highly trained consultants. Writing building applications on its sensor network remains largely intractable as it involves extensive help from an expert in each building's design and operation to identify the sensors of interest and create the associated metadata. This process is repeated for each application development in a particular building, and across different buildings. This results in customized building-specific application queries which are not portable or scalable across buildings.

We present a synthesis technique that learns how to transform a building's primitive sensor metadata to a common namespace by using a small number of examples from an expert, such as the building manager. Once the transformation rules are learned for one building, it can be applied across buildings with a similar metadata structure. This common and understandable namespace can enable analytics applications that do not require apriori building-specific knowledge.

Initial results show that learning the rules to transform 70% of the primitive metadata of two buildings (with completely different metadata structure), comprising 1600 and 2600 sensors, into a common namespace ([13]) took only 21 and 27 examples respectively. The learned rules were able to transform similar primitive metadata in close to 60 other buildings as well, enabling writing of portable applications across these buildings.

[*] Primary Author.

1. INTRODUCTION

Buildings are sites of very large sensor deployments, typically containing up to several thousand sensors reporting physical measurement, continuously. Moreover, with the recent interest in reducing building energy consumption and increasing their efficiency, it is important to consider ways to quickly bootstrap a set of building data streams into an analytical pipeline. These analyses consist of jobs that measure the performance of the building with respect to overall comfort, determine where there are opportunities for energy savings by detecting rooms that drive the energy consumption down – also known and rogue rooms/zones – and finding broken sensors.

Ideally, such analyses could be deployed across many buildings, quickly. However, current 'point' naming conventions and unsystematic recording of metadata form a bottleneck in the scalability of the data integration process. A 'point' refers to a physical location where a sensor is taking measurements. In addition, expanded descriptive information about the point is sometimes unavailable – so determining their meaning is painfully slow or impossible. Because these are conventions carried out by humans, they are inconsistent within and across building data sets. This makes the integration process laborious for building experts and a non-starter for non experts.

If we want to quickly run the job across many sites we need to explore methods automatically normalizing the data and boosting the existing set of metadata per point. This will allow wide *searchability* of points across many buildings at once. In order to meaningfully deal with disparate building streams in a scalable fashion the streams should be *search-*

able across various properties, such as building name, room location, and type. Moreover, we assert that wide searchability is necessary for achieving scalability. By providing a tool for searching across building streams, we minimize the deployment time for applications that allowing them to be used in *all* buildings, not just a single one.

Consider a simple analysis program, which has the ability to identify anomalous readings from a specific kind of sensor. To execute this job, the process organizes each sensor by type and location, organizes a the distribution of readings across them, and identifies broken sensors where some fraction of their readings are above some threshold value on the distribution. The identification step in the process in the perhaps the most challenging because of the problems described. Ideally, the program would search for points the way you search for web pages in a search engine – using semantically meaningful terminology. Some codes and metadata across buildings might be unique but we aim to discover the overlap to order for the search results to yield a higher harvest (increased coverage of the points that meet the search criteria). We can treat both the name and the description as a set of terms that are associated with a measurement point.

Sensor 'point' names (from building-system nomenclature) contains set of codes that are semantically meaningful to the building manager of a specific building. For example, the point `BLDA1R435__ART` is constructed as a concatenation of such codes. The name of the building (first 4 characters), the air handling unit identifier (the fifth character), the room number (R435), and the type ART (area room temperature) – which indicates that this are measurement is produced by a temperature sensor. In addition, to point names there may be some descriptive metadata. The description for this point (if it exists) would describe that this is a "temperature sensors in room 435". However, point names do not always follow the exact same structure within and across buildings and certainly do not follow the same convention across vendors. In this paper we aim to boost the existing metadata by learning the rules of construction through a programming-by-example approach where the user provides some input-output examples to boost the existing metadata with extra terms. We can then search across the boosted metadata to increase our search harvest over time.

We propose a set of techniques which learns how to transform a building's metadata to a common namespace by using a small number of examples from an expert. Once the transformation rules are learnt for one building, it can be applied across buildings with a similar metadata structure. We show how our approach makes it easier to write applications across buildings by demonstrating its use by three different applications: 1) a rogue zone detector and 2) an application that identifies and ranks the most comfortable rooms. We illustrate these on a testbed consisting of nearly 60 buildings comprising more than 16,000 sense points. We also illustrate how this common namespace can help a user write analytics applications that do not require building-specific knowledge and scales across different buildings.

We believe this is an important study given the recent trends in the penetration of the *internet of things* into our homes and environments. Our technique can be used to unify that data across many deployment and enable broad search and exploration of new applications. For example, sensing device names for the internet of things are likely to follow similar conventions with very little context. We argue that unification through boosting will be necessary in this broader domain. Buildings are but one example that serve as a testbed for the proposed techniques.

2. MOTIVATION

Buildings are notoriously complex from a management perspective. They consume a large fraction of the energy produced in the United States and much of is wasted [15]. There has been much work in the building science community to reduce their energy consumption and make them more efficient, but the route to broader impact is typically carried out through regulations guided by the findings of studies in those communities.We aim to let solutions reach buildings *directly* by making sense of the data they produce as quickly and accurately as possible. In order to achieve this at scale, we must explore ways to deal with the data produced from sensors within them and to enable broad analysis across several buildings at a time. Our study focuses on any building equipped with a network of sensors. Nearly three-quarters of commercial buildings contain a rich sensing fabric, installed as part of the building management system [21]. It is the data from these system and variants of it, that we wish to unify and make sense of in a more systematic and automated fashion.

The data problems parallel the complexity of buildings. Adhoc data management practices make it difficult for any analytical solution to be widely ported or run across building systems. When dealing with a small number of points such differences are usually not a problem. Upon visual inspection, encodings are similar enough that the engineer can decode the meaning. However, for automatic processing or processing a large number of points across buildings, these kinds of variations makes it difficult to generalize the character-construction rule set. *Fundamentally, full coverage is attainable if we could learn all the codes and map them to more descriptive search terms.*

At a high level we want to normalize the metadata across all buildings so that one query can run across all of buildings, and we want the normalization process to be automatic, so that the overhead in adding a new building to the set is small – the latter of which is important for achieving scale. Fundamentally, a tradeoff exists between the degree to which we can automate the normalization task and the level of coverage you get in the general rule construction. You can get full coverage with no automation. This is essentially the approach that is common today. Every solution that incorporates a building's data is manually adjusted, specifically for that particular building. You can get some degree of coverage if you use a general set of rules, increasing the automation factor and decrease the manual one. Tuning is quite arduous and in most cases, non-programmers are tuning the data ingestion script.

The problem for a large number of buildings is particularly pernicious to the goal. Although there are some similarities for certain kinds of sensor labels and metadata, searching across them yields results of varying success. Consider the simplest kind of a search, a grep scan across the metadata associated with the points across a set of buildings. If we wish to attain all the temperature sensors or set points for a particular building, without knowing these codes, we should be able to attain it by search for all points in a particular building that measure ”temperature“ or that have “setpoint” in their description.

We demonstrate this by collecting all the metadata across our building testbed. The testbed consists of almost 60 buildings and over 20,000 sense points. We collected the names for each of the set points and any associated metadata that describes the meaning of the name. Then, we run a number of grep searches on the data and calculate the relative success of the query. Figure 1 shows the results for two queries. The ‘temperature‘ grep string is “buildingA room temp” and the ‘setpoint‘ grep string is “buildingA room setpoint”. For the temperature query we attain 87.5% of the actual sense points we are searching for, while for the setpoint query we attain only 56% of the setpoints in the building. We could try different search queries that yield different results, however, because there’s only *some* overlapping metadata terms across buildings, coverage tends to be quite poor.

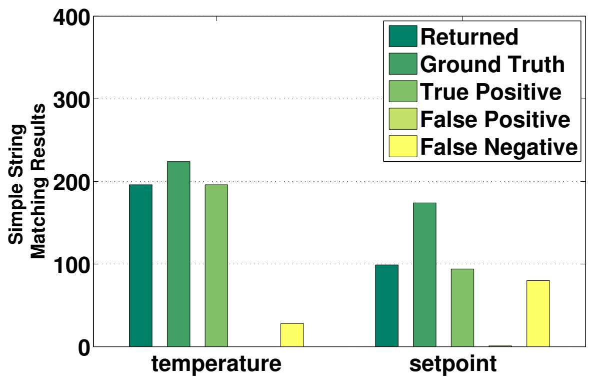


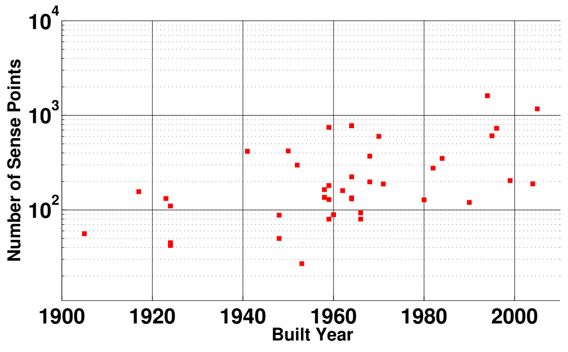
Figure 1: Results when running a grep search on the point names in a single building.

A key observation we have made while attempting to write ingestion scripts in that we create a program P_1 to that parses the data and generates a set of point names n_1 with coverage c_1 of the points we need to obtain. We notice there are points missing, so we write another program P_2 that generates another set of points n_2 which includes some of all of the points in n_1 and gives us coverage c_2. We keep noticing there are points missing and keep either expanding or adjusting the program to get closer and closer to full coverage. In practice, the coverage is not easily attainable without an expert, familiar with the data set, inspecting the results. So the challenge is to attain the good coverage in one (or a few) buildings using input-out examples, using a technique similar to the work by Gulwani et al [9], and use this the resulting program and a few more example in each building to have the process learn the variants of all the codes and boost them with extra tags that are learned from other buildings.

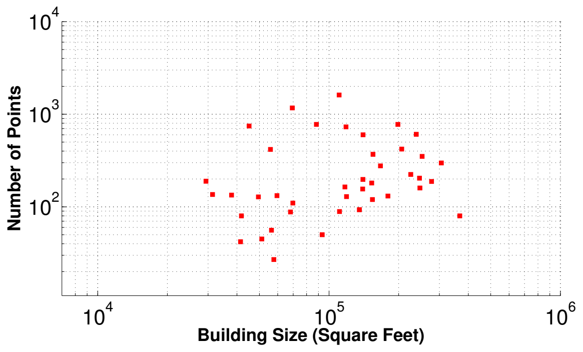
This process allows us to leverage the knowledge of the expert while minimizing the integration overhead per building. Moreover, as more examples the provided, the better that algorithm can learn how to cover a broader set of codes and boost them with common tags. Also, the technique used in previous work is well suited for our problem, since users tend to be non-programmers. The input-output example interactive model is well suited for interacting with non-programmers and getting the kind of information we need from them to automate the program creation process to learn the various point code.

3. BUILDING TESTBED

In our experiments we used an extensive building testbed that consists of 56 buildings containing over 16,000 sense points. These buildings represent a vast range in age, size, and density of deployment. It also represents deployments that were set up by more than one vendor. As expected, newer buildings have many more sense points than older ones – although some old buildings that have been retrofitted have over 1000 points within them. The general trend in the number of sense points versus the year the building is built is roughly linearly increasing in the log of the number of points, as shown in Figure 2a.



(a) Number of Points vs Year Built



(b) Number of Points vs Building Size

Figure 2: Relationship between the built year and the building size on the number of sense points. This data is summarize from a testbed that we used for experiments that consists of almost 60 buildings and over 16,000 sense points.

The maximum number of points in a single building is 1614 and the minimum is 27. The built years spans over 100 years – from 1905 to 2005. The size range spans over an order of magnitude in square footage from about 30,000 square feet to over 360,000 square feet. There is no observable correlation between the size of the building and the number of sense points. Figure 2b shows a log-log plot of the number of points versus the size of the building. There is one large building with many sense points, but the size seems to have little to do with the density of the deployment. The access this this kind of breadth of system and building types makes our study unique.

4. AUTOMATING METADATA ACQUISITION

In this section, we go into detail about how we apply program synthesis techniques and the input-output model of interaction to learn the various semantic labels contained inside a sensor's name[1], in effect boosting the metadata associated with a sensor. We first introduce some basic terminology followed by an overview of the synthesis technique. We then describe how we adapt the technique to the context of sensor name qualification and evaluate our adapted technique on our testbed in Section 5.

4.1 Terminology

The expert is expected to point out *(Tag Name, Tag Value, Value Type)* tuples in the sensor name. A *tag* is mapped on to a substring of the sensor name, which is called its *value.* A tag can have a constant or a variable value. A value should be regarded a *constant* if it is not specific to that particular sensor, and *variable* otherwise.

Sample Input: Suppose the expert is presented with an example `BLDA1R465__ART`. Suppose this sensor name indicates that it is in Building `BLD`, is part of the first air handling unit (ahu), indicated by the character `A1`, in room 465 (`R465`) and it is the area temperature sensor (`ART`). He should qualify it in order as

`BLDA1R465__ART` : (site, `BLD`, const), (ahu, `A`, const), (ahuRef, `1`, var), (zone, `R`, const), (zoneRef, `465`, var), (zone air temp sensor, `ART`, const).

In this example the *site* tag's value is `BLD`, which is not specific to that particular sensor. Hence, the expert should mark it as a constant. On the other hand, the value of the *ZoneRef* tag is specific to that sensor, and hence should be marked as variable.

Sample Output: The synthesis technique should then be able to identify the tags in a new sensor name automatically. For example, given the sensor name `BLDA5R234___ART`, it should output the set of tuples shown below:

`BLDA5R234___ART` : (site, `BLD`), (ahu, `A`), (ahuRef, `5`), (zone, `R`), (zoneRef, `465`), (zone air temp sensor, `ART`).

We term each of these tuples as a *qualification*, because it qualifies a set of alphanumeric characters into normalized metadata tags. We term the output as a *full qualification*, if every alphanumeric character in the sensor name was *qualified* by the set of outputted *tags*, and no extra erroneous tags were applied. This is an input-output example we will refer multiple times to throughout this section.

[1]We use the words *sensor name* and *point name* interchangeably

Tag Names : The goal of the expert should be to use tag names from a normalized building equipment taxonomy schema that has been widely adopted. Currently, there is no consensus schema in the sensor network, or building management system vendor community about a particular schema. Some schemas such as Green Building XML [7], and Industry Foundation Classes [2] require a very high level of detail for every metadata tag, making it unsuitabe for use in our context where that detail might not be available. Instead, we assume that the expert qualifies the sensor names with metadata tags that have been developed as a part of the Project Haystack effort [13], which is an open source effort to develop taxonomies and ontologies for building equipment. This schema is, however, very limited, and not applicable to a wide variety of building-specific sensor points (such as specific alarms, etc). In these cases, we expect the expert to use an easily understandable long-form tag, which is consistent across the entire building. This can be achieved by presenting the expert a set of tags that has already been used in a that building to qualify a particular substring.

4.2 Synthesis technique overview

Transformation Program **P**	:=	if b_1 then e_1
		else if b_2 then e_2
		…..
		else tag not exist in string
Boolean classifier $\mathbf{b_i}$	:=	$d_1 \vee d_2 \vee \ldots \vee d_n$
Conjunct $\mathbf{d_i}$	:=	$p_1 \wedge p_2 \wedge \ldots . \wedge p_n$
Predicate $\mathbf{p_i}$	:=	**Occurs**(v, r, k) | **OccursAtPos**(v, r, c)
Occurs(v, r, k)	:=	True, iff regular expression r occurs in string v , k times
OccursAtPos (v, r, c)	:=	True, iff regular expression r occurs in string v at index c
Extraction Rule **e**	:=	**Substring** (v, p_1, p_2)
Substring(v , p_1, p_2)	:=	Substring of string v between positions p_1 and p_2
Position $\mathbf{p_i}$	:=	**Constant**(k) | **PrecedeSucceed**(r_1, r_2, c) | **ConstantWidth**(k)
PrecedeSucceed(r_1, r_2,c)	:=	Index at the c^{th} intersection of regular expression r_1 and r_2
ConstantWidth(k)	:=	Index $p_1 + k$, where p_1 is starting index of substring
Regular Expression **r**	:=	**Tokens**(T_1, …. T_n)
Token **T**	:=	Alphabets| Numeric | specialToken | ε
		| constant tag value entered by expert

Figure 3: Language for learning substring extraction

We will first describe the high-level logic of the synthesis technique[2]. We will then describe the basic constructs of our synthesis language before explaining some of the intricacies which make it robust.

The Algorithm:

The main aim of the technique is to learn two sets of information from the given input-output examples — (a) which string transformation is applicable on a particular input to produce the output, and (b) what is the set of regular expressions that transform the input string to the output string.

From each user-provided input-output example, the set of all expressions from the language (shown in Figure 3), that could extract the required output string from that input is computed. If there are multiple input-output examples, the substring extraction rules of the multiple examples are intersected to obtain a more concise set of expressions. If the substring extraction rules cannot be intersected, they are maintained as two disjoint sets, which we shall hereby term as a *partitions.*

Finally, for each disjoint set of extraction rules/regular expressions, a boolean classifier is built in the Disjunctive Nor-

[2]similar to [9]

mal Form (DNF), to differentiate the examples in one partition to examples in all other partitions. When a new string is given to this tool, the classifier is applied on it to figure out which partition the new input falls into, and the corresponding set of transformation expressions are then applied on it.

The intuition is that we can independently consider each (*tag*) to be a potential output for an inputed string sensor name. If the tag can be applied to qualify a substring of the sensor name, then the output would be the required substring extraction program. In all other cases, the output would be ϵ or the null string.

The Language:

The top level expression of the language is the classifier — the *If* b_i *Then* e_i structure, which applies the substring expression e_i to the input only if it matches the boolean expression b_i. The boolean function is in DNF form and is composed of predicates of the form $\textbf{Occurs}(v_i, r, k)$, which evaluates to true, iff the input v_i has k occurrences of the regular expression r, or $\textbf{OccursAtPos}(v_i, r, c)$ which evaluates to true iff the input v_i has a regular expression r which occurs at index c.

The Substring expression $\textbf{SubString}(v_i, p_1, p_2)$, evaluates to the substring between positions p_1 and p_2 of the string v_i. $\textbf{Constant}(k)$ denotes the integer position k in the substring. A position expression $\textbf{PrecedeSucceed}(r_1, r_2, c)$ when applied on a string s evaluates to an integer position t in the subject string s such that r_1 matches some suffix $s[0..t]$ and r_2 matches some prefix of $s[t...l]$ (where $l = \text{Length}(s)$). Also, t is the cth such match starting from the left end of the string. If such an position t does not exist in the string, this operator fails. The regular expressions are either just a single token τ, or a token sequence, $\textbf{Tokens}(\tau_1..\tau_n)$, or ϵ (which matches the empty string). The tokens τ comprise of a single token to denote alphabetic characters (referred to as *AlphTok*) , one for numeric characters (referred to as *NumTok*), one for each special character, and one for each constant tag value entered by the user. The output is obtained by applying the resultant $\textbf{SubString}(v_i, p_1, p_2)$ operation.

We provide a couple of examples to elucidate how this technique works.

Example 1:

Consider, again, the sensor name `BLDA1R465__ART`. If the desired output is the substring `ART` can be obtained, among other expressions, by either of the following language transformations : SubString(s, Constant(11), Consant(14)), or SubString(s, PrecedeSucceed(UnderscoreToken, *AlphTok*,1), PrecedeSuccede(*AlphTok*,ϵ, 1)).

Example 2:

Suppose the synthesis algorithm has seen two examples (a) `BLDA1R465__ART`, whose desired output is `A` at index 3, and (b) `BLD__R479_ART`, whose desired output is `479`. One of the possible expressions that the synthesis algorithm can come up with is : **If** b_1 **Then** e_1 **Else if** b_2 **Then** e_2, where $b_1 =$ Occurs(s, *AlphTok UnderscoreTok*,1), $e_1 =$ Substring(s, Constant(3), Pos(*AlphTok*, *NumTok*,1)), and $b_2 =$ Occurs(s, *UnderscoreTok AlphTok*, 2) and $e_2 =$ SubString (s, Constant(6), Constant(9)).

In general, there can be many expressions in the defined language that can obtain the desired substring, and provide a classifier to specify which types of inputs each type of substring extraction should work on.

Thus, for each expert-given input-output example[3] and for each tag in the output of that example, we can compute the set of all expressions from the language that could extract the required tag's substring from that sensor name. We then learn a boolean classifier $b_1$such that all tags that are present in the example evaluate their **If** b_1 **Then** e_1 condition on this example to True, and all the tags that are not, evaluate theirs to be False. If False is returned, then the substring extracted is ϵ, signifying that the tag is not applicable on the sensor name.

If the same tag is present in multiple examples, the tag's new substring extraction expression set is the intersection of its substring extraction expression sets for each of those examples. This may result in disjoint *partitions* of rules for a particular tag. Finally, a boolean classifier is built to differentiate examples of each partition of a tag from all other provided examples.

4.3 Language Intricacies

Challenges

Synthesis techniques in our context face certain challenges. First, the number of tags required to fully qualify an entire building might be large, whereas certain tags may be applicable on a very limited number of sensor names. In such a case, the classifiers and the corresponding regular expressions r should be expressive enough to differentiate a small group of sensor names from the remaining. One way to increase the expressive power of the regular expressions r is to have certain building-specific tokens in addition to the normal alphabetic, numeric and special character tokens. Different conventions of sensor naming from building to building precludes us from having an a priori set of special tokens.

Second, a building may have 1000s of sensor points, making visual inspection of correctness of sensor name qualification very hard. Hence, wrong application of a tag to a particular sensor name is likely to go unnoticed. We mitigate this by being more conservative with the boolean classifiers b.

Below, we list four techniques that augment the basic language shown in Figure 3 to enable correct classification of

4.3.1 Token Set

There are various intricacies in our language to make it robust in the face of unstructured and noisy data. We con-

[3]which in our case is the list of tuples defined in the terminology

ducted an experiment where we provided examples for sensor names from a building containing 1585 sense points, applying the spreadsheet transformation technique in [9]. The synthesis algorithm applies the tags that it has seen from its existing set of input-output examples on the remaining sensor names. After every run of the algorithm, on the present set of input-output examples, a new example was chosen at random from the corpus of all sensor names, and a full qualification of it was provided as the next input-output example. Figure 4 shows the result of our experiment.

We would expect the number of sensor names to have been *fully qualified* to increase with each added input-output example. We find this trend up to around 25 examples, after which the synthesis technique started applying erroneous tags to sensor names. At closer inspection, it turns out that the tokens token set used by the regular expressions and the **Match** predicates were not expressive enough to capture the difference of applicability of different tags.

To illustrate the problem, consider the examples

Example 1 : `BLDA4S1831_STA` : [(site, `BLD`, const), (ahu, `A`, const), (ahuRef,`4`, var), (supply fan,`S`, const), (supply fanRef,`1831`, var), (status point,`STA`,const)] ; and

Example 2: `BLDA3R5871_VAV` : [(site, `BLD`, const), (ahu, `A`, const), (ahuRef, `3`, var), (zone, `R`, const), (zoneRef, `5871`, var), (vav, `VAV`, const)]

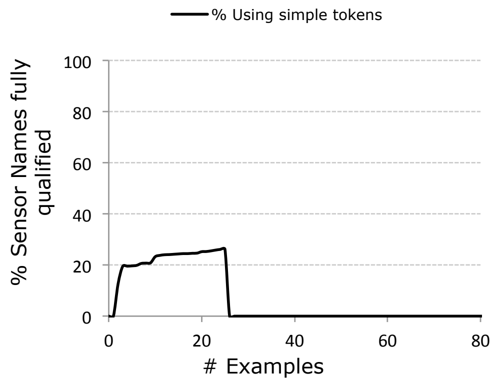


Figure 4: Number of sensor names correctly fully qualified with a token set consisting of *AlphTok*, *NumTok*, *SpecialCharToken*

Both these sensor names have the exact same arrangement of numeric and alphabetic characters, and special symbols, and no regular expression comprising only alphanumeric and special characters would be able to discern between the two. This resulted in erroneous extra tags being applied to sensor names. Hence, we modify the existing language and set of tokens to stay general enough, yet be more expressive. Also note that the set of tokens which make sense in the context of sensor names vary across buildings, and across vendors.

To solve this, we utilize the constant tag values in the examples provided by the expert as special tokens for the regular expressions to be generated from that example, in addition to the normal tokens described above. Thus, the sensor name `BLDA1R465__ART`[4] is treated as a set of tokens [(BLD)(A)[*NumTok*](R)[*NumTok*]+ *UnderscoreTok UnderscoreTok* (ART)].

Similarly, suppose the input-output example provided by an expert for a sensor name `BLD2.PWR.CL42A.REACTIVE POWER` is [(site,`BLD2`,const), (power meter, `PWR`,const), (power meterRef,`CL42A`,var), (reactive power, `REACTIVE POWER`, const)] This string is treated as a combination of tokens [(BLD2) *dotTok* (PWR) *dotTok* [*AlphTok*]+[*NumTok*]+[*AlphTok*] *dotTok* (REACTIVE POWER)].

Note that this provides enough expressibility for the regular expressions to differentiate between the two inputs `BLDA4S1831_STA`, and `BLDA3R5871_VAV` from the user, which was impossible using only the normal tokens.

4.3.2 Splitting constant tags

In order to improve the efficiency of the classifier, we consider a tag having multiple constant values, as different tags. For instance, in our test building, an *exhaust fan* tag was applicable either as the constant characters `E` or the constant character `EF`. Note that the index of the various other tag's values in a sensor name change depending on which of the constant value appears, rendering previously learnt Constant(k) operators useless.

We treat the same tags which maybe represented by two different constant string to be two different tags altogether. This enables a richer intersection set when tag outputs from the expert-given examples are combined.

4.3.3 Boolean Classifiers

Since manual inspection of the qualification of all the points is not feasible, we take steps to strengthen the boolean classifiers corresponding to each tag. We implement different strategies for tags that have constant and variable values.

If a tag has a constant value, we ensure that each boolean clause d in the DNF expression used by the **If Then Else** operator has a **Occurs**(s, constTagValue, k) as the first conjunct p_1 , where k is the number of times the constant value appears in the examples. This ensures that if a tag is evaluated to be applicable on a string, the constant substring does exist in the string.

Next, in cases where a tag which has a variable value, and is a reference to another tag with a constant value which is applicable on the same string, we mandate that the constant tag also have been deemed applicable on the same sensor name. For instance, in the example `BLDA1465_ART`, the tag *zoneRef* whose value is to `465` has a reference to the *zone* tag, which has a constant value. Thus, we would only evaluate the **Substring** expressions for *zoneRef* iff the *zone* tag has been deemed applicable on the same point.

[4]whose output example provided by the expert was (site, `BLD`, const), (ahu, `A`, const), (ahuRef,`1`, var), (zone,`R`, const), (zoneRef, `465`, var), (zone air temp sensor, `ART`, const).

4.3.4 *Considering position of extracted tags*

To make the classifiers more general, so that regular expressions learnt for one building may be applicable to sensor names of other buildings with a similar naming convention, where similar tags occur at the same indices, we include the contruct **OccursAtPos**(v_i, r, k) which evaluates to true iff the regular expression is satisfied at position equal to k.

Suppose, from the technique encounters only one example `BLDA1R465__ART` , and for the *zone temp sensor* tag learns a classifier b_1 = **OccursAtPos**(s,(ART),1) for application of the *zone temp sensor* tag. Now, if another building was abbreviated as `ART`, and the synthesis technique encountered a sensor name of the form `ARTA2R354__ART`, the classifier would fail to apply the *zone temp sensor* tag because there are two matches to the regular expression (ART) in the string, and **OccursAtPos**(s,(ART),1) would evaluate to false. Whenever possible, we apply **OccursAtPos** before **Occurs** expressions while generating classifiers.

5. EVALUATION OF LEARNING BY EXAMPLE

In this section, we gauge the effectiveness of our learning by example technique by evaluating the number of examples required to qualify labels in two large commercial buildings.

5.1 Testbed

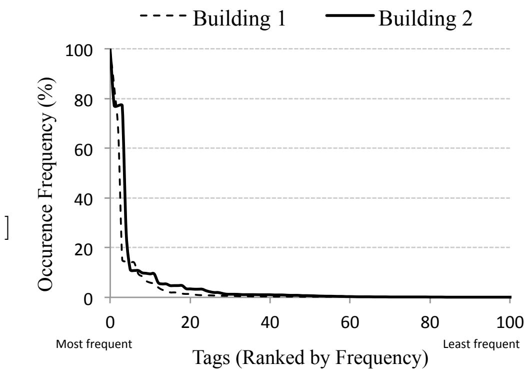
We manually generated ground truth data for all the points in two buildings whose building management system was installed by different vendors. Building 1 has 1586 sensor points and was built in the 1990s. Building 2 was built in the 2000s and has 2551 sense points. The label characteristics of the two buildings are shown in Figure 5.

Figure 5a shows that in these two buildings, a few tags (about 20 in each building) frequently appear in a lot of sensor names. This is pretty common in commercial buildings, where a majority of the points are related to zone or room information. For instance, Building 1, has a room setpoint sensor, an airflow sensor and temperature sensor for each of its more than 200 rooms. For each of these points, the *zone* and *zoneRef* tags are applicable because there exist characters which specify that it is a zone and it zone number. These most frequent tags also fully qualify a large number of the sensor names in both buildings. As shown in Figure 5b, learning proper classifiers and qualifications for about 20 labels could yield a full qualification for 70-80% of the sensor names in both these buildings.

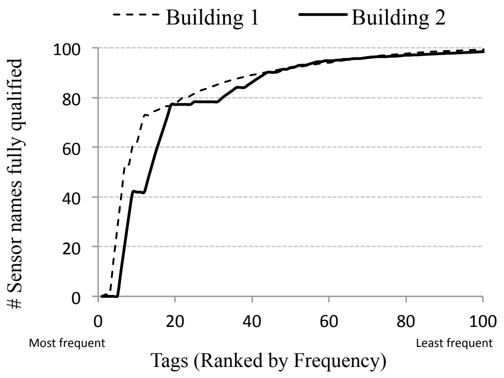
The distribution frequency of applicable tags also has a long tail. These comprise of tags for building specific sensors, alarms or status variables. Thus, one of the main objectives of the learning algorithm is that it does not learn wrong classifiers for tags based on sensor names that fall in this long tail.

5.2 Choosing the Next Example

The large number of sensors in a building pose a challenge in selecting the next example to present to the expert. First, the expert might not always be able to browse through all sensor points to check correct qualification. Also, an expert might visually not be able to discern which points would add the most amount of information to the learning process.



(a) Percentage of sensor names each tag appears in. The x-axis is sorted according to the frequency of occurrence of a label



(b) Percentage of sensor names fully qualified by the highest ranking tags. A point (x,y) indicates that y sensor names could be fully qualified by using labels ranked 1 ... x

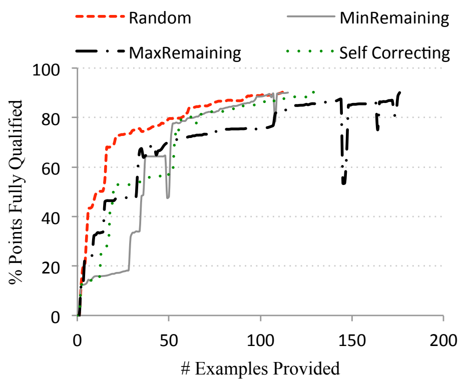
Figure 5: Characteristics of tag application from two buildings we generated complete ground-truth data for, to test our learning technique

We implemented four different generators to evaluate which example should be provided next to the expert:

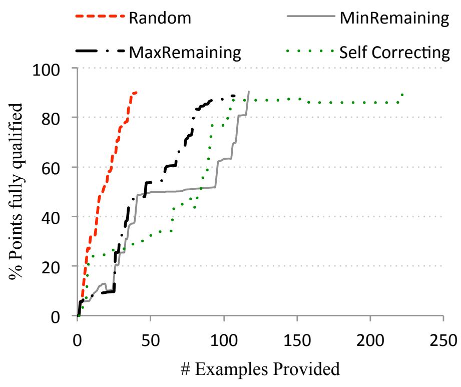
Random: This generator just finds at random the next example to present to the expert. While choosing the example, the random algorithm chooses among the set of sensor names which it feels it has not been able to fully qualify.

MinRemaining : This generator chooses the example, that according to our tool, has the minimum string length left to qualify. The intuition behind this is to gain more concrete knowledge about a small number of labels.

MaxRemaining : This chooses the example, that the learning technique feels has the maximum string length left to qualify. These examples would help the learning technique gain coverage over the space of unseen labels. The more labels the learning technique knows, the more sensor name



(a) Building 1



(b) Building 2

Figure 6: The number of examples required to fully qualify 90% of sensor names in two buildings. The *Random* generator achieves 70% full sensor name qualification within 25 examples for Building 1 and 27 examples for Building 2.

information it will be able to qualify.

Self Correcting : There are some sensor names where the learning algorithm can itself figure out that it has incorrectly qualified a sensor name. There can be three such indicators. First, for a sensor name which has matched its boolean classifier, but none of its set of **SubString** regular expressions is applicable. Second, if the sensor name has been qualified with labels that overlap over the same substring region. Third, the learning algorithm can get a notion of the qualification uncertainty of a sensor name, if its (tag, value) tuples change drastically when a new example has been added. This generator gives the expert the examples that satisfy the most number of these three criteria. Once, none of the points satisfy these criteria, this generator defaults to the MinRemaining generator.

Experiment :

We wrote a script that automatically gave the synthesis tool the example that it asked for, and compared the qualifications for all sensor names outputted by it after the processing the example, to the ground truth. We ran one experiment for each of the generators. We terminated when the number of correct full sensor qualifications reached 90%. A full qualification of a sensor name into (tag, value) tuples is correct if (a) the correct tags were applied on it and obtained the correct values corresponding to each tag, (b) No extra incorrect tag was applied to the sensor name, and (c) the tags were able to fully qualify every alphanumeric character of the sensor name.

Results :

Figures 6 show the results of the four generators on the two buildings. The *Random* generator took the least number of examples to achieve full qualification of 70% of the sensor names, achieving it much quicker than the others. The reason for this is due to the long tail of the label distribution of tag names (Figure 5a). The top 20 most occurring tags, by themselves, can fully qualify about the majority of the sensor names. A random generator has a high probability of finding one of these points, thus acquainting itself more quickly of the most frequently occurring labels. Neither of the other three classifiers is able to achieve that. They get stuck trying to learn regular expressions from sensors with obscure tags (*MinRemaining*), or trying to cover more labels by first qualifying sensors which have been least qualified, which comprise mainly of sensor names which are inconsistently named (*MaxRemaining*), or by choosing form the set of ill-formed sensor names, which would indicate errors to the learning algorithm (*Self-Correcting*).

The number of example reach 90% qualification, is however, similar across all the generators. Going from 70% to 90% takes about 85 extra examples for the *Random* generator. The process is much smoother for Building 2 which has better defined point names, and fewer inconsistent or incomplete point names.

Conlusion :

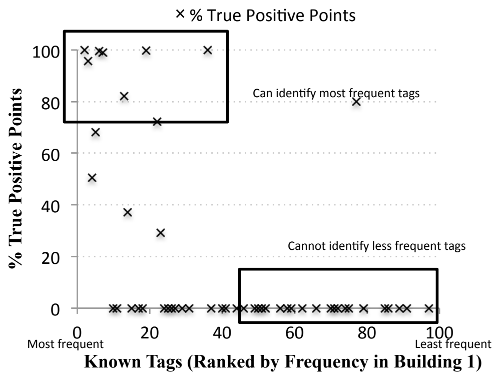
The results show that a *Random* approach while seeking the next sensor name to get an example of, leads the synthesis technique to quickly accurately fully qualify a large fraction of the sensor names (70-80%). This result may also be generalizable to other commercial buildings for which zone-related sensors comprise a major fraction. However, going from 70% full qualification to 90% full qualification takes a long time, as these point comprise of tags that are infrequent.

5.3 Applying Learnt Expressions to unknown buildings

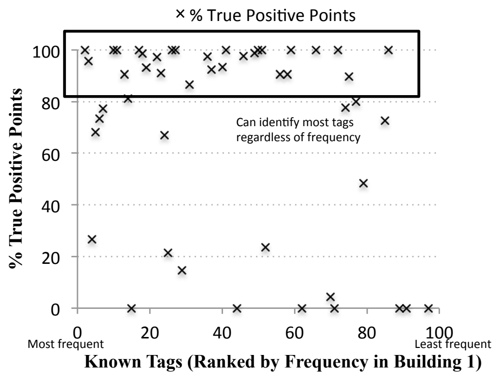
One of the major goals of learning from examples provided by an expert is that the learnt regular expressions may be applicable to other buildings which have similar naming conventions. In this section, we evaluate the efficacy of the regular expressions learnt on Building 1 of our testbed with the remaining 55 buildings on campus, each of whose building management system sensors was commissioned by the same vendor. As mentioned before, the remaining buildings comprise about 16,000 sensor points.

Testbed for experiment:

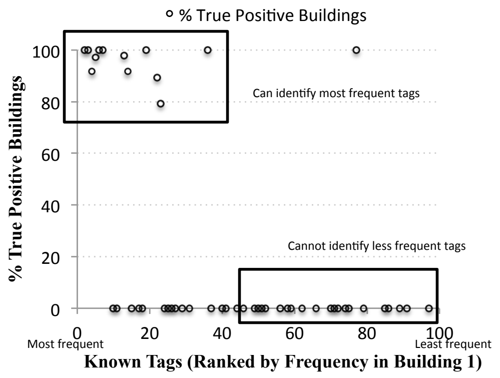
Figure 7 shows the applicability of tags learnt from Building 1 to the remaining buildings. Tags which were most frequent



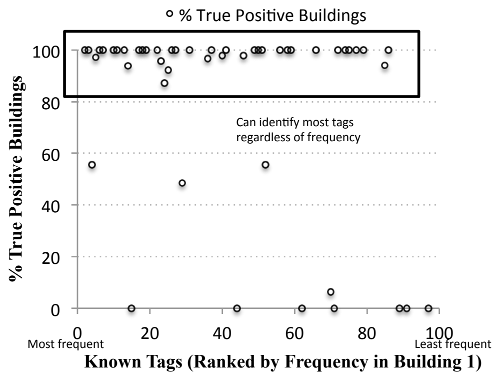
(a) Applying Regular expressions obtained after 50% full qualification of Building 1 by **Random**



(b) Applying Regular expressions obtained after 90% full qualification of Building 1 by **Random**



(c) Applying Regular expressions obtained after 50% full qualification of Building 1 by **Random**



(d) Applying Regular expressions obtained after 90% full qualification of Building 1 by **Random**

Figure 8: Figure showing that while the *Random* generator can quickly obtain full qualification of 70% of a particular building's point names, it is less effective at learning the regular expressions for lower frequency tags, which may also be applicable across a lot of Buildings (Figure 7b). However, if a lot more examples are given, such that the *Random* generator can fully qualify 90% of Building 1's point names (about 110), it get obtain good regular expression for the infrequent tags as well, and be able to correctly apply them in an unknown building.

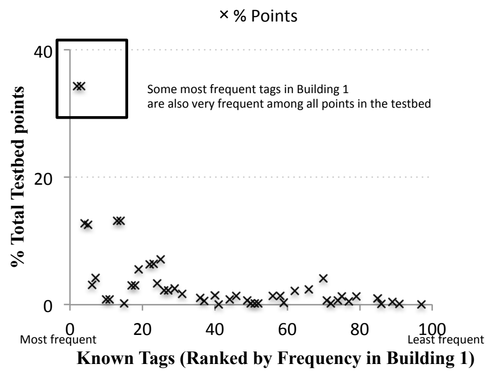
in Building 1 such as *zone, zoneRef* and *ahuRef*, were also widely applicable throughout other buildings. This is expected because all these buildings are commercial offices, with a large number of zone, and a set of points associated with each zone. There are also a few tags which are infrequent in a particular building, but applicable across all buildings, e.g the *return water temp* tag or the *outside air temp* tag. Thus, fully qualifying even one building has the potential to yield regular expressions that can then be propagated across buildings.

Experiment: It was impractical to ground truth all the buildings and its 16,000 sensor points for our experiment. Hence, we hand wrote regular expressions to the best of our efforts to qualify sensor names in those buildings. It is this manual qualification data that we treat as ground truth in the our experiment. We run three experiments, where we use the regular expressions obtained after 50 and 90% full qualification results obtained by the *Random* process in the previous experiment(see Figure 6a).While running these expressions on a different building, we just replaced the token corresponding to the tag *site* to the building-specific *site* token.

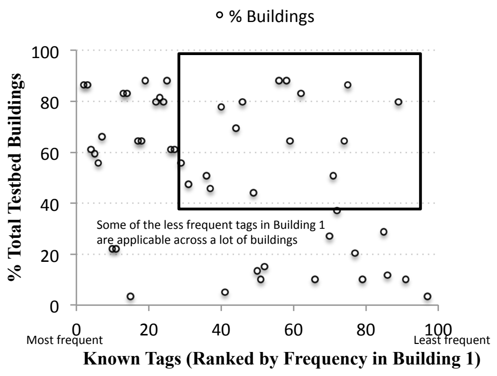
Results:

Figure 8 shows the true-positive rate of the tags applied to the 15000 sensor names across 55 buildings, based for each tag based on our ground truth data. The true positive rate indicates that the required tag was correctly applied where it was applicable, and it was able to qualify the correct substring from the corresponding sensor name.

There is direct effort-applicability correlation. The more fully qualified a particular building is, the more tag applicability expressions are learnt. When enough examples were provided such that Building 1 had correctly fully qualified only 50% of the sensor names, the synthesis technique had either not encountered a lot of the lesser frequent tags, or did not have enough examples to build a robust classifier, leaving the regular expression set unable to discover the same tags in the remaining point names(Figure 8a). Also, this trend was true across buildings, where the inferior set of regular expressions was not able to properly apply the lower



(a) Occurrence frequency of tags learnt from Building 1 to the other 55 buildings in our testbed. The most common tags in Building 1 (e.g *zoneRef*) are also applicable to around 35% of all the sensor in the testbed.



(b) Occurrence frequency of a tag across buildings. Even though some tags are less frequent (e.g *Chilled water temp* tag), they do occur across 60-80% of the buildings.

Figure 7: Applicability of existing tags learnt from Building 1 to the remaining 55 buildings in out testbed. Missing values corresponding to an x-axis value indicates that the particular tag did not appear in any of the other buildings

frequency tags on any building (Figure 8c).

However, when enough examples for 90% full qualification was provided (110 examples for Building 1), the synthesis algorithm was able to correctly identify and qualify a lot of the tags which had a low frequency in Building 1. This is due to the exhaustive list of regular expressions it had to generate to fully qualify the obscure point names in Building 1 in order to get a 90% accurate full qualification.

Some of the tags could not be applicable to other buildings, because they use different constant values for the same tag name. For instance, the *exhaust fan* tag is specified in certain buildings as `EF`, whereas in Building 1 it was specified as `E`. Also, certain tags which extract variable values (the *zoneRef* tag) face some challenges when scaling across buildings.

Conclusion: We conclude that using the *Random* generator to ask for the next example, the number of examples required to qualify a large fraction (e.g 70%) of a commercial building's point names might be few (24 example for Building 1). During the process, though, the generator does not learn the expressions needed to qualify the lower frequency tags, a lot of which are common across buildings.

Trying to accurate fully qualify 90% of a building's point names forces the synthesis algorithm to encounter a encounter the less frequent tag names ,and hence it is able to apply these lower frequency tags when they apply across buildings. However, Building 1 in our testbed required about 85 more examples to reach from a full qualification of 70% of the sensor to a full qualification of 90% of the sensors.

6. CASE STUDY

In this section, we demonstrate that with the metadata automatically expanded and normalizedusing the techniques in Section 4, we are able to implement applications that are generalizable from one building to another building without modification. As a proof of concept, we implement two applications on the two building as test bed: a) identify uncomfortable rooms and b) detect rogue rooms. We also evaluate the metadata expansion technique in terms of the accuracy for both applications compared against the ground truth.

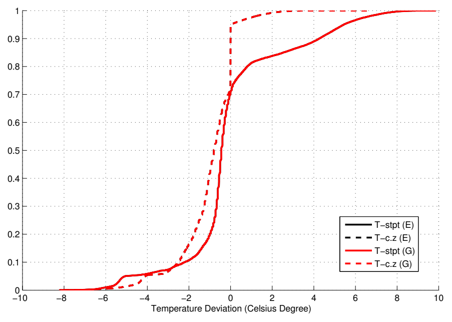
6.1 Experimental Setup

We implement two applications and perform the analysis on the same two buildings used in Section 5.1, and each building is installed with a different management system. Building 1 uses the system from Barrington while building 2 is installed with the Siemens BACnet system [1]. We used the temperature data as well as setpoint information of the rooms in each building. The temperature measurements are reported every 15 seconds and the data used for analysis is from one week in June 2009 and January 2012 respectively. Particularly, we pick the data during the working hours from 9am to 5pm for analysis.

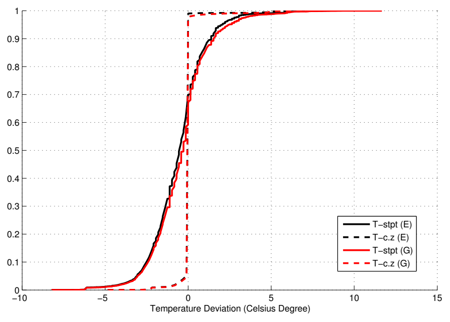
6.2 Uncomfortable Rooms

It's not unusual to have rooms in a building stay extremely cold or hot thus making the occupants feel uncomfortable and incurring energy waste. The discomfort is usually caused by improper setpoint configuration or dysfunction of the HVAC systems, and being able to identify these uncomfortable zones or rooms in the building is vital to improve occupant comfort as well as achieve potential energy savings. With the metadata normalized using our techniques, we are able to search for the desired streams, e.g., the temperature and setpoint of a room, and analyze the thermal performance of different buildings despite of the different naming schema used to label sensors and meters.

To identify the discomfort in a building, for each room we are particularly interested in 1) how much does the temperature deviate from the comfort range? 2) how much does the temperature deviate from the setpoint? To answer both questions, we first search through the points in each building for distinct temperature stream of each room and the corresponding setpoint. Then we compare the temperature

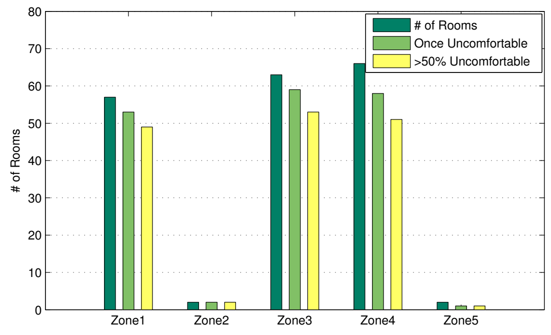


(a) Building 1

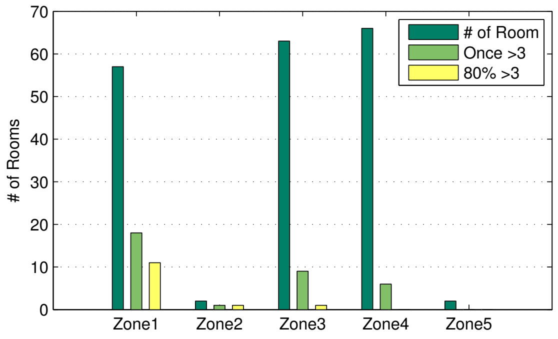


(b) Building 2

Figure 9: For each building, the distribution describes the temperature deviation between: a) room temperature and the corresponding setpoint (solid), b) room temperature and the comfort range suggested by ASHRAE (dashed). The estimated distribution (labeled as "E") based on the expanded metadata and the ground truth distribution (labeled as "G") are both plotted. Note, the estimated ones overlap with the ground truth ones in the left graph.



(a) Uncomfortable Rooms



(b) Rogue Rooms

Figure 10: Breakdown of the uncomfortable and rogue rooms in building 1 by air handler unit zone. Uncomfortable rooms are those whose temperature at least once exceeds the comfort range suggested by ASHRAE. Rogue rooms are those whose temperature deviates from the setpoint more than 3 Celsius degree. For each zone, we show the number of rooms in it, the number of zones at least once meets the criterion, and the number of rooms that meet the criterion in more than 50%/80% of the one-week period.

with the setpoint as well as the suggested comfort range by ASHRAE (73F-81F for summer and 67F-76F for winter) to compute the temperature deviations from the aforementioned three different perspectives in one-week period. We accumulate the results from all the rooms per building and generate the distribution as illustrated in Figure 9.

Figure 9 presents the temperature deviation distribution for both buildings where the results are generated from the expanded metadata following the above steps. We also present the ground truth of temperature deviation distribution, where we manually find the temperature and setpoint streams for all rooms. Each graph shows how much the temperature of a building deviates from the setpoint (solid) and the comfort range (dashed). On average, both buildings are uncomfortable to some degree and to better understand which dominant rooms are uncomfortable, and the estimated distributions using expanded metadata are close to the ground truth ones. To gain further insight, we rank the rooms in each building by how much time they deviate from the comfort zone and the ranking results are shown in Table 1.

Moreover, for building 1 we group the identified uncomfortable rooms down to their corresponding air handler units (AHU), as shown in Figure 10a. For each AHU zone, we present the number of rooms in it, the number of rooms whose temperature have been at least once outside the comfort range thus being uncomfortable, and , the number of rooms that have been uncomfortable in more than 50% of the one-week time. We see that for each AHU zone a large portion of the rooms are uncomfortable in even more than 50% of the time, indicating the building was very likely to operate under a improper schedule. The ground truth analysis covers all the rooms in each building, so all potential uncomfortable rooms are identified here. However, the analysis using the name points expanded with our techniques would miss some of the uncomfortable rooms because the expansion contains certain error rates. We will discuss the error rates later.

6.3 Rogue Rooms

Heating and cooling contribute to the largest portion of energy consumption of a building, and often, HVAC system

*Bldg*1		*Bldg*2	
room#	%	room#	%
326	1	330B	1
340	1	213	1
352	1	148	0.72
364	1	629	0.54
376A	1	768	0.49
380	1	458	0.45
384	1	621	0.44
405A	1	750	0.42
405B	1	571	0.35
410A	1	548	0.29

Table 1: Ground truth for how much time each room's temperature is outside the comfort range: rooms in each buidling are ranked by how much time they are uncomfortable throughout the one week period, and the first ten rooms on the ranking of each building are listed.

operates abnormally either because the system fails itself or the schedule of the building is problematic. And there are often some zones and rooms in a building that are constantly cold or hot than the neighbors thus incurring energy waste. We demonstrated the temperature deviation distribution above, and we are particularly interested in the periods when a room deviates from the setpoint more than 3 Celsius degree, which indicates that the room is highly likely to be under either heating or cooling. Therefore, for each building, using this criterion, we zoom in to the interested portion on the temperature deviation distribution and find rooms falling into this portion in most of the time. The ground truth results are summarized in Table 2. Again, we group the rooms according to their air handler unit ID and show the results in Figure 10b. We see that there are 13 rogues rooms all together in building 1, and 11 of them belong to AHU1, suggesting the unit might be either wrongly configured or misoperating.

*Bldg*1		*Bldg*2	
room#	%	room#	%
330B	1	330B	1
340	1	213	1
420A	1	148	0.93
420	1	768	0.67
698	1	371	0.62
442	0.996	458	0.62
398	0.981	538	0.57
336	0.96	413	0.54
183	0.92	558	0.48
498	0.91	548	0.47

Table 2: Ground truth for how much time each room's temperature deviates from the setpoint more than 3 Celsius degree. For each building, the first column is room number and the second column is the percentage of the one-week time that the room deviates that much. The first ten rooms on the ranking of each building are listed.

6.4 Miss Rate

The metadata expansion can contains certain errors in it therefore when we do a search over the expanded metadata we might not get all the desired streams for analysis. Figure 11 shows the error rates of the search results over expanded metadata for the two test bed buildings. On the left, 50% of the points in building 1 are correctly fully expanded, and doing the two searches "room temperature" and "room temperature setpoint" will get us all desired streams (232 temperature and 243 setpoint). Therefore, performing the uncomfortable and rogue rooms analysis will not miss any rooms in this case. Meanwhile, for building 2 on the right, when 50% of the points are correctly fully expanded, we missed 14 out of 176 for temperature and 27 our of 304 for setpoint, for the same two searches as done on building 1. Since some of the temperature and setpoint streams are not recalled, we would miss some of the uncomfortable and rogue rooms as a result. We also perform another set of experiments where we have 70% of the points in each building correctly fully expanded, but the results are the same as those of 50% expanded case.

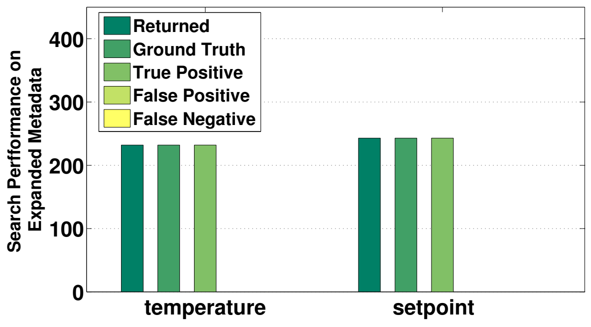
	Bldg 1	Bldg 2
Uncmft	156/0/8	4/0/0
Rogue	13/0/3	3/0/3

Table 3: The number of missed rooms for the two applications for the two test bed buildings. In each cell, we show the ground truth number of rooms/the number of rooms missed by analysis on metadata expansion/the number of rooms missed by a simple grep.

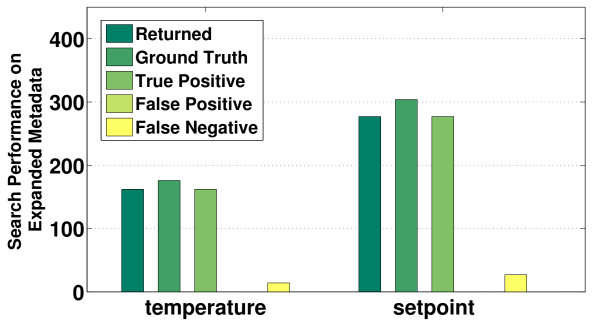
We show the miss rates of the two applications when using the expanded metadata and using a simple grep as a baseline. In each cell of Table 3, the three numbers are for the ground truth number of rooms, the number of rooms missed by running the application on expanded metadata, and the number of rooms missed by running the application on the grep results. We see that, even though the expansion has errors in it, we are still able to find most of the problematic rooms that are otherwise difficult to identify. We conclude that with the expanded and normalized metadata of a building, we can run useful analysis and identify potential problems in it.

7. RELATED WORK

There has been much prior work on metadata generation for videos [6], pictures [17], business audios [22] and taxonomy expansion [23]. Our work is focused on building sensor data and borrows from from the literature in the context of providing a way to search through data from sensors in buildings. Our approach consists of two main components: metadata boosting and search. Our main focus in this paper is on metadata boosting. We use the technique introduced by Gulwani et al.[9] in order to learn the expanded form of 'point' tags. In their work, and related extensions [12, 18, 19, 11, 16, 10], they provide a set of libraries that implement algorithms to learn the various patterns for strings in excel spreadsheets. Because many excel users are not programmers, it is clear that the interaction model should be based on having the user provide input-output examples and for a the system to iteratively learn the pattern the user would write if they were more technical inclined. In our work we use a similar approach and make non-trivial extensions to



(a) Building 1



(b) Building 2

Figure 11: The error rates of searches over the expanded metadata using our techniques. Two searches are performed particularly: "room temp" and "room temp setpoint".

it in order to boost the existing metadata with a normalized set of tags. This allow users to broadly search across all points for multiple buildings at once, without having to engage in the tedious pattern adjustment task for expansion.

The de facto approach in buildings is to form communities that follow are particular standard. Each of the vendors follows their own general structure in point naming, however there are evolving standards that unify these across vendors, such as the Building Information Model (BIM) [20] and Green Building XML [7]. These are used to codify an object structure to describe the various physical components of the building. For example, there are objects for the internal subsystems, the walls, the construction if the windows, and sensors within the building as well, among other things. Each Building software vendor extends or modifies their own BIM version. These standards are mainly for use by architectural design firms to construct and share building models across software suites. We could potentially make use of BIMs in incorporating the descriptive names/tags as extra metadata. We leave this exercise for future work.

Google is incorporating 3D models of buildings and embedding their into Google Earth 3D Buildings offering [8], making them searchable according to their metadata and location. The effort does not include a granularity down to the sensors, but with the recent acquisition of NEST [14], indexing the metadata that describes the physical measurement points within buildings may come next. Our approach is to normalize the metadata for the sensors, so that we can maximize coverage and do analysis across many buildings at once. Because standards are not followed, boosting is critical to achieving normalization and maximizing coverage.

A different approach entiring are those taken by open standards such as BACnet and LonTalk [5, 1] and more recent approaches to describe the sensors more systematically, such as sMAP [3], HomeOS [4] and Building Depot [24]. From a metadata perspective, they essentially bypass the normalization issue. We use them as metadata sources but we address a fundamental data integration problem to achieve wider coverage, faster.

8. CONCLUSION AND FUTURE WORK

In order to meaningfully deal with disparate building streams in a scalable fashion the streams should be *searchable* across various properties, such as building name, room location, and type. Searchability is necessary for achieving scalability. By providing a tool for searching across building streams, we minimize the deployment time for applications that allowing them to be used in *all* buildings, not just a single one. We describe how a set of programming by example techniques can be used to learn how to transform a building's metadata to a common namespace by using a small number of examples from an expert.

In order to adapt synthesis techniques presented in prior work [9] we have to overcome three fundamental challenges: 1) attaining full tag coverage for a building is difficult and the number of tags necessary to attain full coverage is very large. 2) merging the substring rules for each tag is nontrivial and 3) because there are so many points, visual inspection of correctness is very hard. Our adaptation partially overcomes these challenges and we show how the tag expansion results can be applied across many building.

For future work we look to integrate standard feature extraction techniques to enrich the metadata with semantically descriptive terms that can be used to search for points of interest based on deeper attributes embedded in the data itself. Moreover, we plan to integrate more buildings into the metadata suite to cover a large fraction of building and enable wide development of analytics and applications. We also look to explore how our current technique could be applied in the wider internet of things context, as more sensors streams are deployed in the home environment. We believe that such metadata boosting and term indexing technique are necessary to make sense of the explosion of data coming from sensors. Buildings present a major challenge and surely the solutions in this space can be applied in other domains.

9. REFERENCES

[1] American Society of Heating, Refrigerating and Air-Conditioning Engineers. ASHRAE Standard 135-1995: BACnet. ASHRAE, Inc., 1995.

[2] I. F. Classes. Industry foundation classes. `http://www.ifcwiki.org/index.php/Main_Page`.

[3] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA, 2010. ACM.

[4] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, San Jose, CA, 2012. USENIX.

[5] Echelon Corporation. LonTalk Protocol Specification. Echelon Corp. 1994.

[6] K. Filippova and K. B. Hall. Improved video categorization from text metadata and user comments. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 835–842, New York, NY, USA, 2011. ACM.

[7] GBXML. Green building xml. `http://www.gbxml.org/`.

[8] Google. Google earth 3d buildings. `http://www.google.com/earth/explore/showcase/3dbuildings.html`.

[9] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[10] S. Gulwani. Synthesis from examples. *WAMBSE (Workshop on Advances in Model-Based Software Engineering) Special Issue, Infosys Labs Briefings*, 10(2), 2012. Invited talk paper.

[11] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. In *In Communications of the ACM*, 2012.

[12] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 317–328, New York, NY, USA, 2011. ACM.

[13] P. Haystack. Project haystack. `http://project-haystack.org/`.

[14] NEST. Nest labs. `http://www.nest.com`.

[15] Next10. Untapped Potential of Commericial Buildings: Energy Use and Emissions, 2010.

[16] D. Perelman, S. Gulwani, T. Ball, and D. Grossman. Type-directed completion of partial expressions. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 275–286, New York, NY, USA, 2012. ACM.

[17] C. Qin, X. Bao, R. Roy Choudhury, and S. Nelakuditi. Tagsense: A smartphone-based approach to automatic image tagging. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 1–14, New York, NY, USA, 2011. ACM.

[18] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *Proc. VLDB Endow.*, 5(8):740–751, Apr. 2012.

[19] R. Singh and S. Gulwani. Synthesizing number transformations from input-output examples. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV'12, pages 634–651, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] D. Smith. An introduction to building information modeling. *Journal of Building Information Modeling*, pages 12–14, November 2007.

[21] U.S. Environmental Protection Agency. Buildings Energy Data Book, 2010.

[22] H. Wang, D. Lymberopoulos, and J. Liu. Local business ambience characterization through mobile audio sensing. In *Proceedings of the 23th International Conference Companion on World Wide Web*, WWW '14, pages 237–240, New York, NY, USA, 2014. ACM.

[23] J. Wang, C. Kang, Y. Chang, and J. Han. A hierarchical dirichlet model for taxonomy expansion for search engines. In *Proceedings of the 23th International Conference Companion on World Wide Web*, WWW '14, pages 237–240, New York, NY, USA, 2014. ACM.

[24] T. Weng, A. Nwokafor, and Y. Agarwal. Buildingdepot 2.0: An integrated management system for building analysis and control. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 7:1–7:8, New York, NY, USA, 2013. ACM.