A Learning Method for Improving Quality of Service Infrastructure Management in New Technical Support Groups

David Loewenstern[1], Florian Pinel[1], Larisa Shwartz[1], Maíra Gatti[2], and Ricardo Herrmann[2]

[1] IBM TJ Watson Research Center
Hawthorne, NY 10532 USA
{davidloe,pinel,lshwart}@us.ibm.com
[2] IBM Research - Brazil
São Paulo, SP, 04007-900 Brazil
{mairacg,rhermann}@br.ibm.com

Abstract. Service infrastructure management requires the matching of tasks to technicians with a variety of expert knowledge in different areas. Most Service Delivery organizations do not have a consistent view of the evolution of the technician skills because in a dynamic environment the creation and maintenance of a skill model is a difficult task, especially in light of privacy regulations, changing service catalogs and worker turnover. In addition, as services expand, new technical support groups for the same type of services are created and also new technicians may be added, either into a new group or into existing groups. To tackle this problem we evolve a method for ranking technicians on their expected performance according to their suitability for receiving the assignment of a service request. This method makes use of similarities between the technicians and previous tasks performed by them. We propose a strategy for incorporating new technicians and delivery team reorganizations into the method and we present experimental results demonstrating the efficacy of the strategy. Applying this strategy to new teams yields on average acceptable accuracy within 4 hours, though with a wide variation across teams for the first 12 hours. Accuracy and its variability approach the quality of accuracy on older teams over 24 hours.

Keywords: service management, service quality, machine learning, ticket dispatching, request fulfillment

1 Introduction

Composition of atomic services for building more complex and useful services has in recent years become a popular approach to delivering customer defined services. The paradigm of composing and arranging atomic services into complex services is generally a bottom-up approach. From a providers perspective, bottom-up composition of atomic services into complex services is important,

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 599–606, 2012.

particularly with regards to cost-effectiveness. Overall process costs can be reduced by late binding to a service supplier depending on the performance of the service suppliers preceding this step. Cost of staff significantly outweighs infrastructure cost, and the difference is increasing. A large portion of service delivery costs is associated with human effort. It is no longer common to have dedicated technical support groups for a particular type of service or even a customer. To minimize overhead and benefit from economy of scale, service providers use cross-account support groups with skills in a specific technical area or layer.

In this paper these technical support groups are known as pools. A typical pool could support for example UNIX platforms, database services or application services. Customers requests are managed as tickets. Tickets are routed to pools through an initial evaluation outside the scope of this paper. Each pool contains a dispatcher who routes tasks (tickets or work orders) to appropriate technicians within the pool.As the number of industries that utilize computing services grows, the volume of information needed by supporting staff has become great and will continue to increase. Although information technology tools have become indispensable, we continue to rely heavily on human experts for problem resolution requiring deep understanding of existing services and their underlying technology. In a sample study it was found that the most qualified technician resolved an issue better (with respect to SLA metrics) than other people in 75% of cases. Dispatching service requests to the best technician takes a significant fraction of the time required to process the request [1].

Because of these factors, dispatching within service delivery is a good candidate for software to assist the manual process or automate it completely. Skills and availability of these resources are an important property of the service, and ideally they need to be included in the service definition or the service delivery system. In actual practice, however, most organizations do not have a consistent view of this information, due to privacy concerns and volatile nature of this information. Some existing work [2] describes a method for finding an appropriate technician to work on tasks by making use of similarities between the technicians and previous tasks performed by them. As services expand, new pools for the same type of services are created. This means that there are pools with no historical data on pool performance, even though some of the technicians could have some history from their assignments in pools they were part of previously. In addition, new technicians may be added, either into a new pool or into existing pools. The central problem to be addressed by this paper is therefore how to rank a set of new technicians in a new pool according to their suitability for receiving the assignment of a request.

This paper focuses on quickly making use of historical data as it becomes available. Our approach starts by examining what the method in [2] builds in lieu of an explicit skill model. The method builds a model composed of a weighted set of features in which neither the features nor the weights encode any information about individual technicians or their skills. Instead, the weights model the judgment of a dispatcher or group of dispatchers (in a service line) about what features of any technician history and of a request are relevant to determining

how to assign each request in that pool to a technician. Because the features are not specific to a particular pool, but only to the technician history, reassigning existing technicians to existing pools is not an issue. The problem of handling new pools can then be reduced to two smaller problems: how to make use of existing weights from the new pool service line until there is enough data from the new pool to calculate weights normally, and how to determine features for technicians with no history in any pool.

Our results show that for mature pools, with a lookback of 1 day, the technician chosen by the dispatcher was in the top quarter of the predicted ranked list 90% of the time. For new pools the results show that most pools processing smaller numbers of requests show greater variation for the first 12 hours, but this narrows after 12 hours. Average differences are good by the fourth hour, and converge toward zero over the course of the run.

The remainder of this paper has this structure: Section 2 presents related work. Section 3 discusses our method in some detail. Section 4 describes the experiments used to validate the method and presents the results of the experiments. Finally, Section 5 discusses conclusions and future work.

2 Related Work

Related to the work presented in this paper, [3] reviews staff scheduling and rostering problems, and the methods reported in the literature for their solution. More recently, the workflow community has been emphasizing the use of machine learning mechanisms. [4] presents an approach to automatically suggest staff assignment for activities in a workflow. Using assignments of previously completed activities as features, a number of supervised machine learning algorithms are applied to the workflow event log and compared to achieve the best accuracy. [5] is a complementary work proposed as a solution to allocate the most proficient set of employees for a whole business process based on workflow event logs. Furthermore, in [6], a staff assignment decision tree is built for a given workflow activity using a skill data model; assignment rules are then derived from the trees. By contrast, the service requests we work with are individual tasks, and we cannot rely on previously completed workflow activities to build feature vectors. As explained in Section 1, maintaining a skill model is also impossible.

In the IT Service Management domain, [7] examines problem tickets that need to be routed among various expert groups. The authors analyze the contents of incoming tickets to identify a set of semantically relevant past tickets, and then create a weighted Markov model from the resolution sequences of these tickets to generate routing sequence recommendations. This is different from our work, as we want to assign a single resolver to each problem.

In [2], the authors present a solution for the problem of assigning an individual technician in the absence of an explicit skill model. Suitability for assignment of a new work order to a technician is inferred by taking into account the similarity of the work order to previous assignments and the outcomes of the previous

assignments (such as whether the work order had to be reassigned). The measure itself is composed of dynamic work order features calculated from a work order and a continually updating history, and weights computed from historical data using Support Vector Machine (SVM)^{rank} [8].When applied to the steady state problem of existing pools of technicians with sufficiently long histories of prior assignments, the technician that has actually been assigned to the work order by the human dispatcher is ranked among the top quarter of candidate technicians by the algorithm, the Top-Quarter Percentage (TQP), 90% of the time. The following section details methods for modifying the solution for non-steady state cases, with new technicians with no history of prior assignments for calculating feature vectors, or new pools with no training set for inferring the feature weights.

3 Method

Let us call the algorithm presented in [2] as described in Section 2 the **base** algorithm. In a linear ranking SVM such as **base**, ranking is performed by evaluating the target function $F(\boldsymbol{x}) = \boldsymbol{\omega} \cdot \boldsymbol{x} - b$ where $\boldsymbol{\omega}$ is a weight vector, $\boldsymbol{x}$ is a feature vector, and b is a bias. In the technician assignment problem, $\boldsymbol{x}(u, r)$ represents the features of a technician w_u with respect to a request r and is defined by $f_{i,j,k}(w_u, r)$, for all i, j, k, and $F(\boldsymbol{x}(u))$ is a real-valued score such that $F(\boldsymbol{x}(u, r)) > F(\boldsymbol{x}(v, r))$ when technician w_u should be ranked better than w_v for handling request r. Because the target functions are only meaningful when compared with each other, we can ignore the bias and normalize the weights so that $\boldsymbol{\omega} \cdot \boldsymbol{\omega} = 1$. Doing this allows us to mix target functions so long as they are defined on the same set of features.

In the case of a new pool, we cannot calculate $F(\boldsymbol{x})$ initially since there is no training data. However, we can substitute another target function, $F'(\boldsymbol{x}) = \boldsymbol{\omega'} \cdot \boldsymbol{x} - b'$ that had been calculated from some data we do have. Reasonable sources of data for calculating F' include all of the other pools in the same service line as the new pool, or even all other pools processing the same types of service requests. One would expect using F' in place of F would reduce the accuracy of the proposed technician assignments to a degree depending on how similar the new pool is to the pools used to calculate F'. Once the new pool has processed a sufficient number of service requests, it will become possible to calculate F for the new pool.

We would like a smooth transition from F' to F as the new pool processes requests and so generates training data. We propose a **mixture** algorithm, mixing F' and F over a "breaking-in" period. During this period, we use for the new pool a mixed target function $F_{mix}(\boldsymbol{x}) = (\Delta\boldsymbol{\omega'} + (1 - \Delta)\boldsymbol{\omega}) \cdot \boldsymbol{x}$, where $\Delta = (t - t_0)/(t_B - t_0)$, t is the time the request is assigned, t_0 is the time the new pool commenced operations, and t_B is the time the new pool is considered to have finished its breaking-in period. In practice, it is not necessary to recalculate F_{mix} for every request if the pools used to train F' are similar to the new pool.

We define the **hourly** and **daily** methods as implementations of the **mixture** algorithm, with $F_{mix}(\boldsymbol{x})$ recalculated in one hour segments. The alternative weights $\boldsymbol{\omega'}$ are calculated over all other pools in the same service line as the new pool. In **daily** the $\boldsymbol{\omega'}$ are calculated once, from the 24 hours before the start of the experiment, while in **hourly** they are recalculated at the beginning of each one hour segment from the previous 24 hours.

New technicians pose a more difficult problem than new pools do. The features of **base** are calculated from requests previously handled by a given technician; in effect, the feature vector $\boldsymbol{x}(u, r)$ represents the recent work history of technician w_u with respect to a new request r. If w_u is a new technician then $\boldsymbol{x}(u, r)$ cannot be calculated at all. If we could match w_u to some existing technician w_v based on static information about their capabilities, we could use a mixture method similar to the one discussed in the previous section, creating a $\boldsymbol{x}_{\text{mix}}(u, r)$ from $\boldsymbol{x}(u, r)$ and the recent work history $\boldsymbol{x}(v, r)$ of w_v. However, the motivation for **base** is to avoid creating and maintaining such information, so there is no way to match w_u to w_v. In recognition of this, the **mixture** algorithm, like the **base** algorithm, does not recommend technicians without histories. Requests must be assigned to new technicians through some policy outside the scope of this paper, for example according to a mentoring or cross-training policy or on a round-robin basis, until all technicians in the pool have sufficient history. Instead, Section 4 will explore how much history is sufficient.

Another method, **flat**, which extends **base**, is evaluated to gauge the value of using the **mixture** algorithm against a non-mixture alternative. Instead of using a mixture, **flat** uses a flat distribution across all features ($\boldsymbol{\omega'} = \mathbf{1}$) for the first segment and subsequently the unmixed weight vector $\boldsymbol{\omega}$ calculated each segment from whatever data is available in the "new" pool.

It is also common for dispatchers to group together similar requests and assign them to a single technician, a process known as *batching*. For these cases, we devised **batch**, which draws requests from the fifteen minutes prior to the new request, since batched requests appear as a set of individual requests with very similar features (including technician assignment) in a short time span. It's expected that the method works very well with very little history for new technicians: indeed, the ideal amount of history for determining whether to add a given request to a batch is just the time from when its first element was assigned until the time the current element is processed by the dispatcher. Batches are identified by matching all the values from their pool, priority, classification, account and work type features. Technicians are then ranked by the higher number of requests that were assigned to them in the period, if any.

4 Experimental Results

The methods from Section 3 were tested using the data from [2]. To simulate the effect of creating a new pool with new technicians, features and SVM weights were calculated for each pool separately using a fixed protocol:

– Each test ran over the course of 24 hours, divided into 24 hourly segments, with data collected at the end of each segment.

– SVM weights were calculated for each segment using only data dating from the beginning of the day until just prior to the beginning of the segment.
– Features were calculated for each work order using only data dating from the beginning of the day until just prior to the assignment of the work order.

The 24 hour test run was chosen based on the results reported in [2]: although longer look-back windows improved performance, a one day look-back window performed adequately and was used as the baseline for most tests. The data was collected once per hour as a reasonable compromise: more frequent, shorter segments marginally improved accuracy but greatly increased computation time, while less frequent, longer segments obscured differences in accuracy among the methods. Each method was run on seven consecutive days, with the results averaged to smooth out weekly variation in workload.

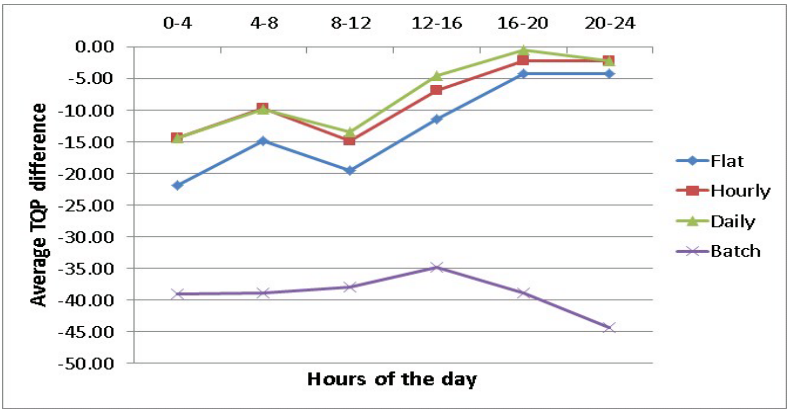

Fig. 1. Average differences in Top-Quarter Percentage (TQP) between several methods applied to "new" pools and **reference**, the reference algorithm applied to the corresponding established pools. See text for details.

Figure 1 presents the results of four separate methods using the same data set as described in the previous section and averaged over 7 one-day runs. Their performance is measured by subtracting their TQP from that of the **reference** method, defined as the **base** algorithm using a 24 hour lookback window ending just prior to the beginning of the run for SVM training and a 24 hour lookback window ending just before each individual work order for feature calculation, thereby treating the same pool as an established pool. The results for each segment then are grouped into 4-hour blocks to simplify the graph.

The **flat**, **hourly** and **daily** methods all start off with lower accuracy than **reference** and asymptotically approach it over the course of each run, with **flat** starting out substantially worse but also converging, indicating both the value of using the **mixture** algorithm and the resilience of the underlying **base** algorithm. Even over the first hours of each run, all three methods perform

fairly well. The relatively good performance of **flat** led to the hypothesis that it and therefore all of the methods exploit batches as described in Section 3. The **batch** method directly tests this hypothesis, demonstrating TQP better than chance but worse than the other methods, indicating the exploitability of batch information but also indicating that batching alone does not explain the success of the other methods.

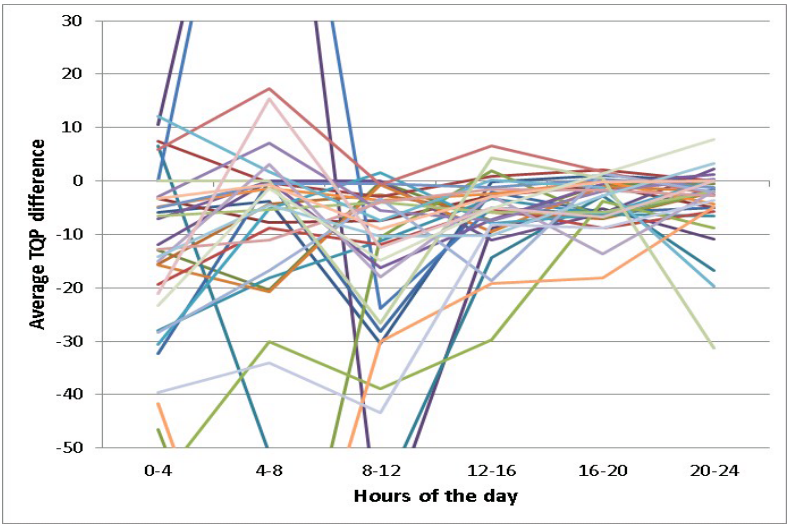

Fig. 2. Average difference in TQP between the **daily** and **reference** methods, broken out per pool. See text for details.

Figure 2 plots the difference between the TQP per pool for the **daily** experiment and the results from the reference method averaged across all runs for pools processing at least 300 requests over the seven days of the experiment (meaning less than two requests per hour). As expected considering the results shown by figure 1, the trend is from lower TQP to higher, with the difference converging on zero over the run. In the first twelve hours, some pools showed wide swings in TQP relative to the reference method; this can be traced to the effect of one or two requests in each pool per run. The number of requests varies over the course of each run, peaking in mid-afternoon (segments 16-20) and falling off at night (segments 0-4 and 20-24); this tends to increase the effect of outlying requests.

5 Discussion

This paper provides a method for ranking technicians on their expected performance by making use of similarities between the assignees and previous tasks performed by them. The central problem that we addressed is how to rank a

set of technicians according to their suitability for receiving the assignment of a request without maintaining an explicit skill model describing which skills are possessed by each technician, in particular for a new pool containing either new technicians or technicians from other pools. Our method builds a model composed of a weighted set of features in which neither the features nor the weights encode any information about individual technicians or their skills. Instead, the weights model the judgment of a dispatcher or group of dispatchers (in a service line) about what features of any technicians history and of a request are relevant to determining proper assignments. Because the features are not specific to a particular pool, but only to the technicians history, reassigning existing technicians to existing pools is not an issue. For mature pools, with a lookback of 1 day, the technician chosen by the dispatcher was in the top quarter of the predicted ranked list 90% of the time. For new pools the results show that most pools processing smaller numbers of requests show greater variation for the first 12 hours, but this narrows after 12 hours. Average differences are good by the 4th hour, and converge toward zero over the course of the run.

In future work we will explore using the amount of available history to adjust the length of the breaking-in period to allow faster convergence where the data permits. We will also look for alternatives to service lines as a method for finding similar pools.

References

1. dos Santos, C.R.P., Granville, L.Z., Cheng, W., Loewenstern, D., Shwartz, L., Anerousis, N.: Performance management and quantitative modeling of IT service processes using mashup patterns. In: Proceedings of the 7th International Conference on Network and Service Management, CNSM 2011 (2011)
2. Loewenstern, D., Pinel, F., Shwartz, L., Gatti, M., Herrmann, R., Cavalcante, V.: A learning feature engineering method for task assignment. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium, NOMS 2012 (2012)
3. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153, 3–27 (2004)
4. Liu, Y., Wang, J., Yang, Y., Sun, J.: A semi-automatic approach for workflow staff assignment. Comput. Ind. 59, 463–476 (2008)
5. Yang, H., Wang, C., Liu, Y., Wang, J.: An Optimal Approach for Workflow Staff Assignment Based on Hidden Markov Models. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2008 Workshops. LNCS, vol. 5333, pp. 24–26. Springer, Heidelberg (2008)
6. Ly, L.T., Rinderle, S., Dadam, P., Reichert, M.: Mining Staff Assignment Rules from Event-Based Data. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 177–190. Springer, Heidelberg (2006)
7. Sun, P., Tao, S., Yan, X., Anerousis, N., Chen, Y.: Content-Aware Resolution Sequence Mining for Ticket Routing. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 243–259. Springer, Heidelberg (2010)
8. Joachims, T.: Training linear svms in linear time. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2006, p. 217 (2006)