Big Data Exploration via Automated Orchestration of Analytic Workflows

Alina Beygelzimer
IBM T. J. Watson Research Center

Anton Riabov
IBM T. J. Watson Research Center

Daby Sow
IBM T. J. Watson Research Center

Deepak S. Turaga
IBM T. J. Watson Research Center

Octavian Udrea
IBM T. J. Watson Research Center

Abstract

Large-scale data exploration using Big Data platforms requires the orchestration of complex analytic workflows composed of atomic analytic components for data selection, feature extraction, modeling and scoring. In this paper, we propose an approach that uses a combination of planning and machine learning to automatically determine the most appropriate data-driven workflows to execute in response to a user-specified objective. We combine this with orchestration mechanisms and automatically deploy, adapt and manage such workflows across Big Data platforms. We present results of this automated exploration in real settings in healthcare.

1 Introduction

With the emergence of multiple Big Data platforms that handle large volumes of streaming and stored data, it is becoming possible to support massive data exploration tasks in many different domains. These domains include cybersecurity, healthcare, financial services, manufacturing process control, as well as several environmental monitoring applications. More specifically, in intensive care, healthcare providers need large-scale and real-time exploration of medical records, test results, and physiological data streams from monitored patients to detect complications as early as possible. There are several ways to explore this data for these detection problems. As a result, such an exploration requires constructing and orchestrating a large number of analytic flows, i.e. workflows composed of atomic analytic components for data selection, feature extraction, modeling, and scoring. The scale of data requires the use of a distributed setting, potentially across multiple compute platforms (e.g. offline learning on Hadoop and online scoring on a real-time stream computing platform). This places a near insurmountable burden on end users and analysts who want to utilize these platforms and analytics in their domain, and motivates the need for autonomic management of the analytic workflows.

In this paper we propose a solution based on autonomic computing principles for creating, deploying, self-managing and adapting analytic workflows in response to an end-user's high level specification of their objectives. Specifically, we propose an approach that combines planning and machine learning to automate the composition and choreography of these workflows in large-scale distributed data exploration tasks. The use of machine learning in autonomic compute systems has been explored previously in limited settings related to scheduling and resource management. In [14] [20] the authors use reinforcement learning for fairly scheduling resources in a large-scale production grid, and resource allocation in distributed settings, respectively. The use of planning in software composition with semantic constraints has been studied in web services [18]. We build on an existing planning-based composition tool, MARIO [15], which was originally created to allow end-users to compose and deploy analytics on multiple platforms.

Combinations of planning and learning have been used in robotics [3] for exploring and partitioning complex state spaces in noisy and stochastic settings, and for imitation learning [16]. However this work is primarily focused on exploring uncertain environments as opposed to analytic workflow composition, selection, and orchestration, as discussed in this paper. Planning and learning for analytic workflow selection has been recently studied in [10]. The authors use a data mining ontology to capture an algorithm's inductive bias, and use learning to select the algorithm to use in different settings. However, this does not consider the problem of constructing analytic workflows by dynamically composing such individual algorithms.

In contrast, in this paper, we focus on both the composition of analytic flows as well as the data-driven dynamic selection of appropriate flows to meet an end-user objective. By describing atomic analytic components with semantic annotations, we provide end users with the ability to specify their objectives using a business relevant semantic vocabulary. This objective is associated with an appropriate objective or loss function that may be used to evaluate performance. We use planning techniques to identify feasible analytic compositions in response to the user-specified goal. We then use online learning to iteratively explore the space of feasible compositions to determine the most appropriate workflows to deploy in a data-driven manner. We combine planning and learning with orchestration mechanisms that allow us to deploy and manage analytic workflows on a large-scale, real-time, distributed stream processing platform (IBM InfoSphere Streams) [19]. This automation allows us to adapt to dynamics in the data, availability of new analytic components, as well as system resource constraints, while providing predictive performance.

This paper is organized as follows. We describe the technical details of our approach and the underlying system architecture in Section 2, including the planning component 2.1 and the learning component 2.2. We then describe results of using this system on real-world exploration problems in a healthcare setting in Section 3 and highlight both predictive performance as well as system dynamics. We finally conclude with a discussion and directions for future work in Section 4.

2 System Description

Our design for an autonomic system for data exploration is based on three observations of the typical application scenario: (i) there are multiple analysis workflows that have different degrees of usefulness for a data analysis objective (and that degree may change over time); (ii) each such workflow is a combination of data sources and analytics, including feature extraction, model building and scoring components; (iii) available computing resources may limit the number of workflows (combinations) that can be in execution at any time.

Consider a healthcare application scenario [1] in which the objective is to predict complications in an ICU setting ahead of time. The available data includes offline data such as histories and outcomes of previous patients, the history of the current patient; slow changing data such as results of physician ordered tests; and live streaming data such as sensor measurements from the patient's monitor units (e.g., ECG, blood oxygen levels, respiration rate, etc.). The system also has available analytics that can extract both simple and complex features from this data, build a variety of machine learning models from this data (including but not limited to decision trees, SVMs, etc.). Compositions of these algorithms into workflows are required to solve the detection problem, however only some workflows are meaningful,

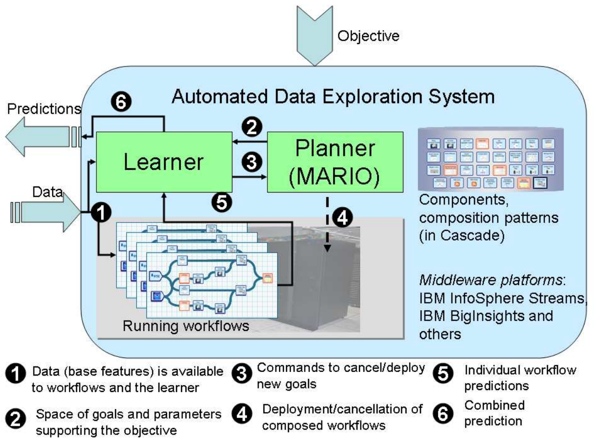
Objective
Automated Data Exploration System
Predictions
Data
Learner
Planner (MARIO)
Components, composition patterns (in Cascade)
Middleware platforms: IBM InfoSphere Streams, IBM BigInsights and others
Running workflows
1 Data (base features) is available to workflows and the learner
2 Space of goals and parameters supporting the objective
3 Commands to cancel/deploy new goals
4 Deployment/cancellation of composed workflows
5 Individual workflow predictions
6 Combined prediction

Figure 1: System architecture

and moreover the choice of workflow depends on context and varies over time. We use planning and learning to dynamically determine the most effective analytic workflow to construct and deploy given our computational resources, and the user specified task.

With these objectives in mind, we propose a design described in Figure 1. There are two main components of the system described in the remainder of this section. The *planner* has access to a repository of descriptions of analytic components and patterns available to the system. The analytic components are semantically annotated, and together with the constraints expressed in the patterns describe the space of analytic workflows that can be automatically composed. The purpose of the planner is to discover the set of goals and parameters that match a specific objective such as predicting patient complications, and for each such goal to automatically compose, generate code and deploy an analytic workflow that realizes the goal. A specific goal relevant in intensive care may be to detect ectopic or abnormal heart beats by observing electrocardiograms (ECG), identifying individual heart beats, extracting spectral features, and classifying them using a decision tree algorithm.

The *learner*'s mission is threefold: (i) over time, it learns the effectiveness of the various analytic workflows deployed for the objective; (ii) it makes a single prediction for specific complications as a function of the effectiveness of the analytic workflows and their individual predictions; (iii) it samples from the space of available analytic workflows that match our computational resources – the sampling is performed as a function of the learned effectiveness of workflows. The learner will continuously re-evaluate the current mix of analytic workflows deployed and, when deciding to change this set, communicate with the planner which will compose and deploy the analytic workflows, potentially on multiple platforms. The learner limits the total load on the system by limiting the number of flows that are active at any given time, and the middleware platforms further ensure efficient allocation of distributed computational resources to the flows selected by the learner.

In our implementation, the planner (including orchestration components) is running as a set of plugins in the Equinox OSGI container, while the learner component is running as a streams processing application on the IBM InfoSphere Streams platform. The two components communicate via a REST API over HTTP. We now describe these components in more detail.

2.1 Planning Component: MARIO

We rely on automated planning to adaptively determine the set of available analytic workflows given changing conditions, resources, data sources, data transforms, and analytics. Any new analytic added to our system is automatically combined with other compatible analytics to generate multiple new workflows. The planner also automatically eliminates inefficient workflows based on estimates of computational cost and reasoning about semantic equivalence of results. The remaining workflows are then made available for the learner to instantiate.

The planning, deployment orchestration, and development environment (for describing workflow composition constraints and semantics of analytics) in our implementation are provided by MARIO (Mashup Automation with Runtime Invocation and Orchestration) [15]. MARIO is responsible for:

1. Generating the complete set of distinct, efficient and valid analytic flows, given the set of analytics, data sources and composition patterns.

2. Generating platform-specific code and deploying individual selected flows with specified parameter values when instructed by the learner.

Our implementation is capable of generating code for IBM InfoSphere Streams and can be extended with plug-in code generators for other Big Data platforms, such as Apache Hadoop. In addition, MARIO provides a web application for end users, allowing them to inspect the results of running flows, predictions made by the learner, and request additional processing to be deployed.

SPPL planner MARIO uses a specialized planner to solve compositions as planning problems described in SPPL [17]. SPPL is derived from the general-purpose domain description language PDDL [8] and includes modifications to improve planning performance in workflow composition applications. In SPPL, description of the planning task includes description of planning domain, consisting of a set of actions with preconditions and effects defined as lists of user-defined predicates, and description of the planning problem containing the predicates of the initial state and the goal state. Given the planning task, the SPPL planner finds an optimal plan, i.e., a partially ordered set of actions that achieve the goal state when applied to the initial state, and optimize a linear objective subject to linear budget constraints.

Cascade composition patterns MARIO generates SPPL descriptions automatically based on composition patterns specified in Cascade [15] that describe composition constraints and software component semantics. Composition constraints are defined by defining a flow graph with points of variability and parameter ranges. Figure 2 includes an example pattern described in Cascade for a simple classification problem we used in experiments. The graph consists of two nodes, transform and classification, with two possible implementations of the transform. Ranges of parameters are specified as enumerations separated by a vertical bar "|". Implementations of *Classification*, *Features_DCT* and *Features_FFT* are defined separately in Cascade by providing platform-specific code fragments. Different choices of parameter values or implementations of the transform can be selected independently, thus generating many possible individual workflows based on the pattern.

In general, Cascade patterns can describe any directed acyclic graphs and can be recursive. For example, *Classification* can be defined as another pattern consisting of lower-level components. Individual components, i.e. analytics or data sources, can be implemented in any programming language supported by MARIO. Since MARIO only generates code for execution on target Big Data platforms and does not process any data itself, it places no restrictions on supported data types or the complexity of analytics. MARIO does not verify schema compatibility between connected components, and Cascade Developers have to ensure that the pattern enforces input/output compatibility of components.

2.2 Learning Component: Learner

At each step, the planner identifies the current set of feasible analytic flows, but it cannot tell which of these flows are useful for the current prediction problem. The goal of the learner is to automatically explore the space of feasible flows, learning the best current combination to deploy, subject to resource constraints. This is achieved in a data-driven way using feedback from the environment.

Our core learning algorithm is online gradient descent with several improvements, including adaptive feature-dependent gradient updates [6, 12] modified for loss nonlinearity as described in [11]. This approach works very

```
composite EctopicBeatDetectionPattern(output o) {
param
 string $NUM_PTS:
  UserParam("Number_Of_DCT_Features","|16|32|48|");
 string $MODELTYPE:
  UserParam("MODELLING_TYPE","|J48|NB|");
 string $MODELTRAINING:
  UserParam("NUMBER_OF_TRAINING_SAMPLES",
            "|500|1000|2000|");
graph
 stream transform =
     Features_DCT(){ param DCT_NUM_PTS: $NUM_PTS; }
   | Features_FFT(){ param FFT_NUM_PTS: $NUM_PTS; }
 stream o = Classification(transform){
   param CLASS_NUM_PTS: $NUM_PTS;
      MODELTYPE: $MODELTYPE;
      MODELTRAINING: $MODELTRAINING;  }
}
```

Figure 2: Example of a Cascade pattern.

well in high dimensional sparse feature spaces, typical in the applications we target.

We now describe the meta-learning problem (learning across other analytics which may themselves use learning), starting with the full information setting, when the learner can run and observe outputs of all available analytic flows at each step. Then we discuss the exploration problem arising from not being able to run most flows at every step due to resource constraints.

The meta-learner operates in an online setting where it repeatedly

1. receives input vector $\mathbf{x}_t \in \mathbb{R}^d$, which includes the outputs of all available analytic flows at time t,

2. makes its prediction $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$ of the target variable $y_t \in \mathbb{R}$, where $\mathbf{w}_t \in \mathbb{R}^d$ is the current linear model.

3. upon receiving feedback y_t, updates

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - 2\eta_t(\hat{y}_t - y_t)\mathbf{x}_t,$$

   where η_t is the learning rate at time t.

While the update rule above assumes the squared loss $\ell(\hat{y},y) = (\hat{y}-y)^2$ commonly used in regression, other loss functions are supported as well. The choice of the loss function is driven by the prediction problem. The gradient update actually used in the system is more complex, based on a combination of improvements described in detail in [6, 12, 11]. Other learning parameters of the learner that govern the form of the gradient, e.g. the learning rate can be optimized on a given problem using progressive validation loss [2, 4].

The analytic flow exploration problem requires extentions to this basic setting. Under resource constraints, when the number of feasible flows is large, they cannot be all instantiated together, hence the learner needs to carefully select which flows to run. Thus instead of observing the entire vector $\mathbf{x}$, the learner can only probe into it sparingly, observing only a small subset of values each time. Such attribute efficient learning for linear regression has recently been explored [5, 9], and we build on these for our exploration. While we omit details here, the intuition derives from the model in the basic setting. When properly normalized to account for the scale of flow outputs, the learned weight vector $\mathbf{w}_t$ indicates the relative importance of each flow. Flows whose weights are close to zero do not have much predictive edge for the current prediction problem. The learner's model $\mathbf{w}_t$ is continuously adapted to changes in the underlying data distribution, as illustrated in Section 3, and this model may be used (e.g. as a probability distribution) to control how the flows are sampled.

There are several other open problems for this exploration. First, we need to account for switching cost considerations associated with starting and stopping workflows. Second, we need a mechanism for learning new useful nonlinearities automatically from data. Finally, while in many applications, the loss function is known (as in classification or regression), there are scenarios, e.g. contextual bandit learning [7] where the loss function has to be learned as well. We are extending our learner component to tackle these open problems, and our results on these extensions will be described in detail in a separate publication.

3 Experimental Results

In this section, we describe two different types of illustrative results – results on a simulated example to highlight the adaptation and convergence of the learning based exploration, and some preliminary results on real-world datasets in healthcare.

3.1 Convergence and Adaptation Results

To illustrate the system's convergence and adaptation, we generated a regression dataset consisting of five features, $x_1,\dots,x_5$, with values drawn independently at random from $[0,1]$ at every step. We then set the label at different time periods as shown in Table 1. We set $\alpha = \frac{t-t_s}{t_e-t_s}$. The experiment includes both sudden as well as gradual variations in the label characteristics. We create 15 analytic flows, that correspond to five self-products, and the ten pairwise products of features $x_1,\dots,x_5$, and use a squared loss function $\ell(\hat{y},y) = (\hat{y}-y)^2$.

Figure 3 shows the corresponding instantaneous squared loss. The best constant's loss in this generated example was 0.1395, with the best constant predictor being 0.1675. The adaptive learner gives a relative improvement of 96.4% in squared loss in this case. Observe

Start(t_s)	**End**(t_e)	y
1	4000	$(x_1 - x_2)^2$
4001	6000	$(x_2 - x_3)^2$
6001	8000	$(1-\alpha)(x_2 - x_3)^2 + \alpha(x_4 - x_5)^2$
8001	10000	$(x_4 - x_5)^2$

Table 1: Generated Data.

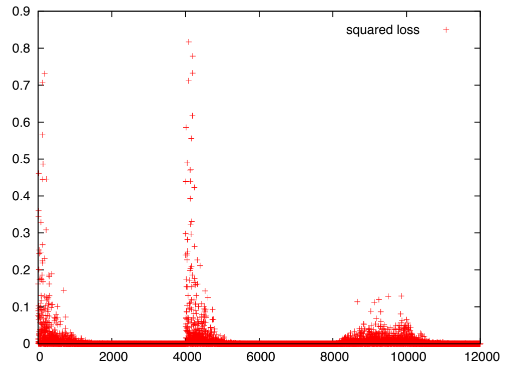

Figure 3: Instantaneous squared loss

the sharp increase in loss in step 4,000 when the target sharply changes to another function. The loss increases smoothly when the target starts to drift away from the learned function in step 8,000.

Figure 4 shows how the weights of different flows evolve in this experiment. The red curve corresponds to flow x_1^2, which is predictive only in the first 4,000 steps, while the target is $x_1^2 - 2x_1x_2 + x_2^2$. Its weight goes down to 0 when it no longer carries any predictive signal. The green curve corresponds to flow x_1x_2; the system learned to use it with coefficient -2 for the first target, and then the coefficient went down to 0 when the target changed to $x_2^2 - 2x_2x_3 + x_3^2$. The coefficients of features $x_1, \ldots, x_5$ are close to 0 for the entire experiment.

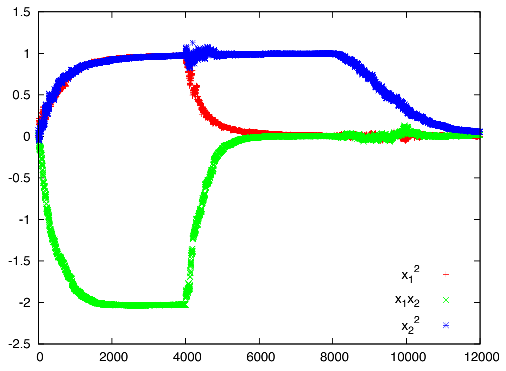

Figure 4: Weight evolution of different analytics

3.2 Exploring Healthcare Data

We consider an exploration problem in healthcare, focused on streaming analysis of Electro Cardiogram (ECG) signals from hospitalized patients. The application is focused on identifying *ectopic* or irregular heartbeats, that are indicative of potential problems to the patient. Detecting such beats from ECG signals may be viewed as a binary classification problem, and requires online learning to adapt to the time-varying nature of the input signals that vary with patient state, medications etc.

We make use of annotated data from the MIT Physiobank [13], a database with around 135000 annotated heartbeats (around 16000 ectopic)- corresponding to raw ECG data from 47 patients. The analytic workflow required for ectopic beat detection includes heartbeat and feature extraction, followed by binary classification. The Cascade pattern is shown in Figure 2. There is a choice between different transformations (DCT or FFT) with retention of only `$NUM_PTS` (16, 32, 64) coefficients, for feature extraction.

As classifiers, we use Weka [21] implementations of Decision Trees (J48) and Naive Bayes (NB) with periodic retraining - controlled by parameter `$MODELTRAINING` (500, 1000, 2000), i.e. retraining of the model is performed every `$MODELTRAINING` heartbeats. Hence, the resulting analytic flow space includes 36 combinations (2 transforms, 2 classifiers, and 3 values each for `$NUM_PTS`, and `$MODELTRAINING`).

These flows are deployed on the IBM InfoSphere Streams processing platform. We deploy a separate data source and feature extraction job for each patient, with a common set of classifier flows for all patients that are selected dynamically by the interaction between the learner and the planner. We deploy these jobs across a cluster with 8 compute nodes with 8 cores each and a shared filesystem.

We compare the prediction performance (in terms of probability of detection p_D and probability of false alarm p_F) of this automated deployment with the best handtuned deployment, achieved after multiple man-hours of experimenting and tuning in batch mode. We also include results to indicate the impact of different resource constraints on the deployment. These are shown in Table 2. For each automated experiment, we were allowed to deploy a maximum of 5 analytic flows at one time. As we can see, the results of automated deployment experiment 1 (A1) are comparable to the best handtuned results - slightly higher p_D and slightly higher p_F – a strong argument for automation. Additionally, by comparing A1 with A2, we observe that performance suffers as fewer resources are available. This is explained by the flow startup time - the time it takes since when the flow is requested by the learner to the time it starts pro-

Test	Nodes	Flow Startup Time	p_D	p_F
HandTuned	8 nodes	–	0.75	0.045
A1	8 nodes	90 sec	0.77	0.05
A2	4 nodes	205 sec	0.72	0.14

Table 2: Prediction Performance.

ducing predictions. In A2, with 4 nodes, most nodes are busy, hence it takes longer for a requested flow to be deployed (on average 205 seconds, during which time the flow misses processing around 240 beats)- and hence the prediction performance suffers. The difference of 0.1 in false alarm rate corresponds to around 12000 more normal beats being labeled ectopic.

We are conducting more detailed validation of these results with a more complex analytic flow space (with larger number of possible flows), with a dynamic change in the available analytics, and with finer grained performance and resource measurements.

4 Conclusion and Next Steps

We present a system that uses combinations of planning and machine learning to automate the orchestration of analytic workflows in Big Data settings. We use planning to identify feasible analytic workflows given descriptions of composition patterns and individual analytical building blocks. We use learning to explore the space of possible workflows and automatically identify appropriate combinations of these flows to deploy in response to dynamically changing data characteristics. We deploy this system to tackle a real-time ectopic beat detection problem in healthcare, and show that the automated system is able to produce results comparable with the best hand-tuned analytics. We are in the process of replicating these results across other domains such as cybersecurity, and using other Big Data platforms such as Hadoop. Interesting directions for future research include the use of hierarchical learning and planning, system resource scheduling and adaptation, and combining these with domain-specific reasoning to exploit domain expertise better.

References

[1] Blount, M., Ebling, M., Eklund, J., James, A., McGregor, C., Percival, N., Smith, K., and Sow, D. Real-time analysis for intensive care: Development and deployment of the artemis analytic system. *IEEE Engineering in Medicine and Biology Magazine 2* (2010), 110–8.

[2] Blum, A., Kalai, A., and Langford, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory (COLT)* (1999), pp. 203–208.

[3] Bulitko, V., and Koenig, S. Planning and learning in a priori unknown or dynamic domains. In *Proceedings of the IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains* (2005).

[4] Cesa-Bianchi, N., and Gentile, C. Improved risk tail bounds for on-line algorithms. In *NIPS* (2005).

[5] Cesa-Bianchi, N., Shalev-Shwartz, S., and Shamir, O. Efficient learning with partially observed attributes. *Journal of Machine Learning Research 12* (2011), 2857–2878.

[6] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12* (2011), 2121–2159.

[7] Dudík, M., Hsu, D., Kale, S., Karampatziakis, N., Langford, J., Reyzin, L., and Zhang, T. Efficient optimal learning for contextual bandits. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI)* (2011), pp. 169–178.

[8] Fox, M., and Long, D. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research 20*, 2003 (2003), 61–124.

[9] Hazan, E., and Koren, T. Linear regression with limited observation. In *Proceedings of the 29th Conference on Machine Learning (ICML)* (2012).

[10] Hilario, M., Kalousis, A., Nguyen, P., and Woznica, A. A data mining ontology for algorithm selection and meta-mining. In *ECML/PKDD Workshop on Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD-09)* (2009).

[11] Karampatziakis, N., and Langford, J. Online importance weight aware updates. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)* (2011), pp. 392–399.

[12] McMahan, H. B., and Streeter, M. J. Adaptive bound optimization for online convex optimization. In *Proceedings of the 23rd Conference on Learning Theory (COLT)* (2010), pp. 244–256.

[13] MIT PhysioBank: Arrhythmia Database. `http://www.physionet.org/physiobank/database/mitdb/`.

[14] Perez, J., Germain-Renaud, C., Keleg, B., and Loomis, C. Utility-based reinforcement learning for reactive grids. In *Proceedings of the Fifth International Conference on Autonomic Computing* (2008), pp. 205–6.

[15] Ranganathan, A., Riabov, A., and Udrea, O. Mashup-based information retrieval for domain experts. In *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), ACM, pp. 711–720.

[16] Ratliff, N., Silver, D., and Bagnell, J. A. D. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots 27* (2009), 25–53.

[17] Riabov, A., and Liu, Z. Scalable planning for distributed stream processing systems. In *Proceedings of ICAPS 2006* (2006).

[18] Sirin, E., Parsia, B., Wu, D., Hendler, J., and Nau, D. HTN planning for Web service composition using SHOP2. *Journal of Web Semantics 1*, 4 (2005), 377–396.

[19] IBM Infosphere Streams. `http://www-01.ibm.com/software/data/infosphere/streams/`.

[20] Tesauro, G., Jong, N., Das, R., and Bennani, M. Hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the Third International Conference on Autonomic Computing* (2006), pp. 65–73.

[21] Weka data mining in Java. `http://www.cs.waikato.ac.nz/ml/weka/`.