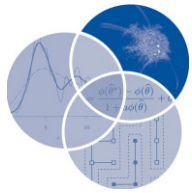
Advanced Review



Algorithm quasi-optimal (AQ) learning

Guido Cervone,[1]* Pasquale Franzese[2] and Allen P. K. Keesee[3]

The algorithm quasi-optimal (AQ) is a powerful machine learning methodology aimed at learning symbolic decision rules from a set of examples and counterexamples. It was first proposed in the late 1960s to solve the Boolean function satisfiability problem and further refined over the following decade to solve the general covering problem. In its newest implementations, it is a powerful but yet little explored methodology for symbolic machine learning classification. It has been applied to solve several problems from different domains, including the generation of individuals within an evolutionary computation framework. The current article introduces the main concepts of the AQ methodology and describes AQ for source detection(AQ4SD), a tailored implementation of the AQ methodology to solve the problem of finding the sources of atmospheric releases using distributed sensor measurements. The AQ4SD program is tested to find the sources of all the releases of the prairie grass field experiment. *WIREs Comp Stat* 2010 2 218–236 DOI: 10.1002/wics.78

Keywords: AQ learning; machine learning classification; evolutionary computation; source detection; atmospheric emissions

The algorithm quasi-optimal (AQ) learning methodology traces its origin to the A^q algorithm for solving general covering problems of high complexity.[1,2] An implementation of the AQ algorithm in combination with the variable-valued logic representation produced the first AQ learning program, AQVAL/1, which pioneered research on general-purpose inductive learning systems.[3] An early application of AQ to soybean disease diagnosis was considered one of the first significant achievements of machine learning.[4]

Subsequent implementations, developed over the span of several decades, added many new features to the original system, and produced a highly versatile learning methodology, able to tackle complex and diverse learning problems. In the current implementation, AQ is a multipurpose machine learning methodology that generates rulesets in attributional calculus. Because of a wide range of features and a highly expressive representation language, recent members of the AQ family of programs are among the most advanced symbolic learning systems.

The rapid development of computer technology and high-level programming languages throughout the 1980s and 1990s prompted researchers to port the original Lisp version of the AQ methodology to new programming environments. These developments, however, were performed in an academic environment and primarily for educational purposes, and they often lacked stability, reliability, and ease of use associated with other commercial or more popular classification programs, such as classification and regression trees (CART)[5,6] and C4.5.[7]

As a result, despite such continuous development, use of AQ programs has been limited, especially outside the main developing group. In addition, limited AQ usage may be the result to some extent of the complexity associated with running various different AQ implementations (i.e., variations based on different parameter settings available within the basic AQ framework). That is, changing modes, tolerance levels, thresholds, and other parameters that can be adjusted in AQ to address different kinds of data or set up AQ to generate different types of output is an exercise in fine tuning, and although correct parameter setting is not overly difficult to learn, it does take some effort and even a modicum of trial and error with specific datasets.

*Correspondence to: gcervone@gmu.edu

[1]Department of Geography and Geoinformation Science, George Mason University, Fairfax, VA 22030, USA

[2]Center for Earth Observing and Space Research, George Mason University, Fairfax, VA 22030, USA

[3]Department of Statistics, George Mason University, Fairfax, VA 22030, USA

DOI: 10.1002/wics.78

This article describes a complete rewrite of the AQ algorithm, specifically tailored for the problem of detecting the source of an atmospheric pollutant release from limited ground sensor measurements. Before undertaking the task of developing an entirely new AQ program, we analyzed relevant available existing AQ implementations, such as AQ15[8] and AQ18.[9] Our analysis indicated that the above implementations were optimized for speed, rather than for the extensibility or comprehensibility of the code. Additionally, many of the features included in the existing implementations were not needed in the context of atmospheric source detection, whereas some crucial features were missing. Therefore, the specialized AQ for source detection (AQ4SD) represents for the AQ approach a fresh new start. Although AQ4SD can be used as a general machine learning classifier, it was specifically designed to work with large noisy datasets containing a limited number of primarily real-valued attributes but up to hundreds of thousands of cases. The main application of AQ4SD is to generate new candidate solutions in a non-Darwinian evolutionary computation process. It was optimized to run iteratively and included a new mechanism for incremental learning to refine previously learned patterns or rules using only new events, rather than starting the learning process from scratch each time. Using knowledge acquired from the analysis of the previous codes, the new design aims at making the new implementation reliable, easy to use, and easy to modify and extend, while retaining relevant features previously implemented in AQ rule learning systems.

The article is structured as follows: the first section gives an introduction to the AQ methodology; it discusses the implementation of AQ4SD, and discusses the strengths and disadvantages; Section *Advantages and Disadvantages of the AQ Methodology* presents advantages and disadvantages associated with the AQ methodology compared with other methods; Section *Evolutionary Computation Guided by AQ* discusses the use of AQ as main engine for an evolutionary computation process; Section *Source Detection of Atmospheric Releases* discusses the problem of source detection of atmospheric releases and presents a technique based on AQ learning to identify the source of unknown releases; and Section *Results* presents the results from the application of AQ to identify the sources of the real-world Prairie Grass experiment.[49] Finally, Section *Discussion* summarizes the main contributions of the articles and the results of the experiments.

AQ METHODOLOGY

Overview

This section reviews the main features of AQ-type learning. A detailed description of various aspects of the methodology can be found in Refs 1–3, and 9–12. AQ pioneered the sequential covering (a.k.a. 'separate and conquer') approach to concept learning. It is based on an algorithm for determining quasi-optimal (optimal or suboptimal) solutions to general covering problems of high complexity.

AQ is a machine learning classifier that generalizes sets of examples with respect to one or more sets of counter-examples. The input data for AQ is therefore made of labeled data, or in other words data which is already assigned to a particular class or group. Unlike clustering, a form of unsupervised learning, whose goal is dividing unlabeled data into distinct classes, AQ is a form of supervised learning, wherein classified data are generalized to identify the characteristics of the entire class.

In its simplest form, given two sets of multivariate descriptions, or events, $P_1, \ldots, P_n$ and $N_1, \ldots, N_m$, AQ finds rules that cover all P examples (a.k.a. positive events) and do not cover any of the N examples (a.k.a. negative events). More generally, each multivariate description is a classified event of type $x_1, \ldots, x_k$, and c, where each x is an attribute value, and c is the class it belongs to. For each class c, AQ considers as positive all the events that belong to class c, and as negative all the events belonging to the other classes.

The algorithm learns from examples (positives) and counterexamples (negatives) patterns (a.k.a. rules) of attribute values that discriminate the characteristics of the positive events with respect to the negative events. Such patterns are generalizations of the individual positive events and depending on AQ's mode of operation may vary from being totally complete (covering all positives) and consistent (not covering any of the negatives) to accepting a tradeoff of coverage to gain simplicity of patterns.

The AQ learning process can proceed in one of two modes: (1) the theory formation (TF) mode and (2) the pattern discovery (PD) mode. The PD mode was introduced in AQ18 and was not part of the original methodology. In the TF mode, AQ learns rules that are complete and consistent with regard to the data. In other words, the learned rules cover all the positive examples and do not cover any of the negative examples. This mode is mainly used when the training data can be assumed to contain no errors. The PD mode is used for determining strong patterns in the data. Such patterns may be partially inconsistent or incomplete with respect to the training data. The PD

mode is particularly useful for mining very large and noisy datasets.

The core of the AQ algorithm is the so-called *star generation*, the process of which can be done in two different ways, depending on the mode of operation (TF or PD). In TF mode, the star generation proceeds by selecting a random positive example (called a seed) and then generalizing it in various ways to create a set of consistent generalizations (that cover the positive example and do not cover any of the negative examples). In PD mode, rules are generated similarly, but the program seeks strong patterns (that may be partially inconsistent) rather than fully consistent rules. This star generation process is repeated until all the positive events are covered. Additionally, when run in PD mode, the generated rules go through an optimization process which aims at generalizing and/or specializing the learned descriptions to simplify the patterns.

AQ4SD Features and Implementation

AQ4SD is, as noted above, a total rewrite of the AQ algorithm, specifically optimized to solve the problem of source detection of atmospheric releases (see Section *Source Detection of Atmospheric Releases*). It shares many parts with the earlier version of AQ20[11] but includes new features and optimization algorithms for the source detection problem. The development of AQ20 was led by the first author in close collaboration with many faculty and student members of the Computer Science Department and Machine Learning Laboratory at George Mason University.

AQ4SD is written in C++ making extensive use of the standard templated library (STL)[13] and generic design patterns.[14] The entire code comprises about 250,000 lines. The goal of the AQ4SD algorithm was to be suitable as a main engine of evolution in a non-Darwinian evolutionary process (see Section *Evolutionary Computation Guided by AQ*) to find the sources of atmospheric releases, using sensor concentration measurements and forward atmospheric transport and dispersion numerical models.

AQ4SD was thus optimized to be used iteratively because evolutionary computation is based on iterative processes. It is tailored primarily toward real-valued (continuous) attributes, and it uses a novel method that does not discretize real-valued attributes into ordinal attributes during preprocessing. It is also optimized to work with noisy data, as sensor concentration measurements often contain errors and missing values. Finally, as sensors are usually very limited in number but record very long time series, AQ4SD is optimized to run with a very large number of events with a small number of attributes. Experiments, for example, were performed with up to 1,000,000 training events, each comprised of 20 real-valued attributes.

Although a formal analysis of the complexity of the AQ algorithm is beyond the scope of this article, experimental runs showed that AQ complexity is polynomial. In particular, it is a low-order polynomial in the number of events and a higher order polynomial in the number of negative events. The lower complexity increases associated with an increase in positive events are due to the fact that in optimization during learning, only uncovered positive events are used to evaluate rules (whereas all negatives, or a sample of all the negatives, are used to evaluate the rules).

The following sections describe the algorithms and data types used and implemented in AQ4SD. Because AQ4SD is an implementation of a general methodology, when *AQ4SD* is specified in the text, it refers to specific features or implementation details of AQ4SD itself, whereas when *AQ* is specified, it refers to concepts and theories that apply to the general AQ methodology.

AQ Events

The AQ input data consists of a sequence of events. An *event* is a vector of values, where each value corresponds to a measurement associated with a particular attribute. An event can be seen as a row in a database, with each value an observation of a particular attribute where columns are the different attributes.

AQ events are a form of labeled data, meaning that they are or can be classified into one of two or more classes. Therefore, each event contains a special attribute *class*, which identifies which class it belongs to. A sequence of events belonging to the same class is called an *eventset*.

Additionally, two different types of events can be used by AQ: *training* and *testing*. Training events are used by AQ to learn rules. Testing events are used to compute the statistical correctness of the learned rules on events not used during learning.

AQ Rules

AQ uses a highly descriptive representation language to represent the learned knowledge. In particular, it uses rules to describe patterns in the data. A prototypical AQ rule is defined in logical Eq. (1).

$$\text{Consequent} \longleftarrow \text{Premise} \sqsubset \text{Exception} \qquad (1)$$

where consequent, premise and exception are conjunctions of conditions. While premise and consequent are mandatory, the exception is optional and used only in very special circumstances. Although exception has been implemented in AQ4SD, it is not being used, because it often leads to over fitting in the presence of very noisy data. A condition is simply a relation between an attribute and a set of values it can take.

$$[\text{Attribute. Relation. Value(s)}] \tag{2}$$

Depending on the attribute type, different relations may exist. For example, for unordered categorical attributes, the relations $<$ or $>$ cannot be used as they are undefined. A complete set of the relations allowed with each attribute type is given in Section *Attribute Types*). Typically, the consequent consists of a single condition, whereas the premise consists of a conjunction of several conditions. Equation (3) shows a sample rule relating a particular cluster to a set of input parameters. The annotations p and n indicate the number of positive and negative events covered by this rule.

$$\begin{aligned}[\text{Cluster} = 1] \longleftarrow\ & [\text{WindDir} = \text{N} \ldots \text{E}] \\ & [\text{WindSpeed} > 10\,\text{m/s}] \\ & [\text{Temp} > 22^{\circ}\text{C}] : p = 11,\ n = 3\end{aligned} \tag{3}$$

This type of rule is usually called *attributional* to be distinguished from more traditional rules that use a simpler representation language. The main difference from traditional rules is that referee (attribute), relation, and reference may include internal disjunctions of attribute values, ranges of values, internal conjunctions of attributes, and other constructs. Such a rich representation language means that very complex concepts can be represented using a compact description. However, attributional rules have the disadvantage of being more prone to over fitting with noisy data.

Multiple rules are learned for each cluster, and are called a *ruleset*. A ruleset for a specific consequent is also called a *cover*. A ruleset is a disjunction of rules, meaning that even if only one rule is satisfied, then the consequent is true. Multiple rules can be satisfied at one time because the learned rules could be intersecting each other. Equation (4) shows a sample ruleset:

$$\begin{aligned}[\text{Cluster} = 1] \longleftarrow\ & [\text{WindDir} = \text{N} \ldots \text{E}] \\ & [\text{WindSpeed} > 10\,\text{m/s}] \\ & [\text{Temp} > 22^{\circ}\text{C}] : p = 11, n = 3 \\ \longleftarrow\ & [\text{WindDir} = \text{E}] \\ & [\text{Date} = \text{July}] : p = 5, n = 0 \\ \longleftarrow\ & [\text{Pressure} > 1010] \\ & [\text{Date} = \text{Sep}] : p = 1, n = 0\end{aligned} \tag{4}$$

Each rule has a different statistical value. Assuming 13 positive events associated with cluster 1, the first rule in Eq. (4) covers not only most positive events in the cluster (11 of the 13 events) but also three negative events. This means that AQ was run in PD mode, to allow inconsistencies to gain simpler rules. The second rule covers less than 50% of the events and the third covers only 1, but both without covering any elements in other clusters. Therefore, there is a tradeoff between completeness, namely the number of events covered out of all the clouds in the cluster and consistency, namely the coverage of events from other clusters.

Attribute Types

AQ4SD allows for four different types of attributes, nominal, linear, integer, and continuous. Each attribute type is associated with specific relations that can be used in rule conditions.

Nominal: Unordered categorical attribute for which a distance metric cannot be defined. Nominal attributes do not naturally or necessarily fall into any particular order or rank, like the colors or blood types or city names. The domain of nominal attributes is thus that of unordered sets. The following relations are allowed in rule conditions: equal ($=$) and not equal ($\neq$).

Linear: Ordinal categorical attribute that is rankable, but not capable of being arithmetically operated upon. Examples of linear attributes are small, medium, large, or good, better, best. Such attributes can be sorted and ranked but cannot be multiplied or subtracted from one other. The following relations are allowed in rule conditions: equal ($=$), not equal ($\neq$), lesser ($<$), greater ($>$), lesser or equal ($\leq$), and greater or equal ($\geq$).

Integer: Ordinal integer-valued attribute without a prefixed discretization and without decimal values. Integer attributes allow only whole numbers, such as 20, or -77. The following relations are allowed in rule conditions: equal

(=), not equal (≠), lesser (<), greater (>), lesser or equal (≤), and greater or equal (≥).

Continuous: Ordinal real-valued attribute without a prefixed discretization but which contains a decimal point and a fractional portion. The following relations are allowed in rule conditions: equal (=), not equal (≠), lesser (<), greater (>), lesser or equal (≤), and greater or equal (≥). Previous versions of AQ dealt with continuous variables by discretizing them into a number of discrete units and then treating them as linear attributes. AQ4SD does not require such discretization as it automatically determines ranges of continuous values for each variable occurrence in a rule during the star generation process.

AQ Algorithm

The AQ learning process can be divided into four different parts: data preparation, rule learning, postprocessing, and optional testing. The following sections address each part individually.

The input data is made of a definition of the attributes (variables), AQ control parameters for each of the four parts mentioned above, and the raw events. The output of AQ consists of the learned rules, which can be displayed in textual or graphical form. Different versions of AQ used different ways to define the format of the input and output. Because the different methods do not affect learning, they are not discussed in this article. AQ4SD uses the input/output format described in Ref 12.

Data Preparation

The AQ learning process starts with data being read from a file (when used as a stand alone classifier) or from memory (when embedded in a larger system). The data is processed by the data preparation mechanism, which checks the data format for correctness, corrects or removes ambiguities, selects the relevant attributes (a.k.a. feature selection), and applies rules for incremental learning.

Some versions of AQ can also automatically or through user input generate new attributes to change the data representation. This feature, called constructive induction, was first implemented in a specialized version of AQ17[15,16] and is not implemented in AQ4SD.

Resolving Ambiguities

An ambiguity is an event that belongs to two or more classes. For the purpose of learning, each event must be unique and belong to a single class. AQ has four different strategies to resolve ambiguities:

Positives: The ambiguous event is kept in the positive class (the class rules are being learned from) and eliminated from all the other classes.

Negatives: The ambiguous event is eliminated from the positive class.

Eliminate: The ambiguous event is eliminated and not used for learning.

Majority: The ambiguous event is associated to the class where it most appears.

Attribute Selection

In general, AQ learns rules to discriminate between classes using only the smallest number of attributes.[a] Therefore, AQ performs an automatic attribute selection during the learning phase, selecting the most relevant attributes, and disregarding those apparently irrelevant. Unfortunately, especially for large noisy problems, irrelevant attributes can lead to generation of incorrect rules. To avoid this problem, AQ can be set to create statistics for each of the attribute values, namely a measure of how many positives and negative examples, respectively, each attribute value covers. AQ can try to keep only those attributes that seem to have more discriminatory information between classes.

This is only a rough approximation, as individual attributes might have little discriminatory information when considered singularly but can help the generation of excellent rules in combination with others. Creating such statistics is a quick linear operation that requires a one-time analysis of the entire data or of a statistical sample of the data. Because such statistics are also used by the learning and optimization algorithms, there is not a significant computational overhead introduced by this attribute selection method.

Rules for Incremental Learning

One of the main advantages of AQ (see Section *Advantages and Disadvantages of the AQ Methodology*) is the ability to refine previously learned rules as new input events become available. The input data can specify rules that describe either a previously learned concept or constraints between attributes. For example, they can specify that a particular combination of attributes cannot appear together in a rule or that the boundaries of the search space are reduced under particular attribute values.

As described in detail in Section *Rule Learning*, AQ starts generating rules by comparing positive

and negative events and keeps specializing previously learned rules with new conditions when they cover negative events. In incremental learning mode, the set of rules being specialized does not start with an empty set but with those specified in the input data. No other aspects of the learning are affected except in the case of extreme ambiguities, when the supplied rules do not include any positive examples. In such situations, AQ cannot use the input rules as it is not able to evaluate their positive and negative coverage.

Rule Learning

This is the core of the AQ methodology, where rules are generated from examples and counterexamples. AQ generates rules by an iterative process aimed at identifying generalizations of the positive examples with respect to the negative examples. Recall that positive examples are those labeled for the target class, and negatives are those belonging to all the other classes.

Algorithm 1 High Level AQ Algorithm

Require: $|P| > 0$ AND $|N| > 0$
1: $P' \Leftarrow P$; $R = 0$
2: **while** $|P'| > 1$ **do**
3: Select random p from P'
4: $r \Leftarrow STAR(p, N, LEF, maxstar)$
5: $P' \Leftarrow P' - [P' \bigcap r]$
6: $R \Leftarrow R + r$
7: **end while**

The main algorithm for AQ is illustrated in Algorithm 1. Although several variants and optimization mechanisms have been developed, the main core shown is true for the main AQ methodology. AQ requires two nonempty eventsets, one of positive and one of negative events, where at least one positive event and one negative event are not ambiguous. The algorithm starts by making a new list of positive events yet to be to cover P'. The algorithm loops until all positives have been covered.

The algorithm starts by selecting a random positive event, called the seed from among P', and then creates a star (Section *Star Generation*) for that example.[b] The result of the star is a rule that generalizes the seed and does not cover any of the negatives (TF mode) or can allow an inconsistent coverage for simpler rules (PD mode).

A lexicographical evaluation functions (LEF) is used to evaluate the rules during the star generation (Section *Lexicographical Evaluation Functions*). Next, all the events covered by rule r are removed from the list P'. Rule r is guaranteed to cover at least the seed but might cover many if not all the events in P'. Rule r is then added to the list of rules R to be added to the final answer.

Star Generation

The central concept of the algorithm is a star, defined as a set of alternative general descriptions (rules) of a particular event (a 'seed') that satisfy given constraints, for example, do not cover negative examples, do not contradict prior knowledge, etc.

Algorithm 2 Star Generation Algorithm

1: $r = 0$
2: **for** all n in N **do**
3: $r' \Leftarrow p \dashv n$
4: $r'' \Leftarrow r' \bigcup r$
5: $r'' \Leftarrow LEF(r'', maxstar)$
6: **if** $\{Mode = PD\}$ **then**
7: **if** $q(r'') - q(r) > minq$ **then**
8: $r = r''$
9: **end if**
10: **else**
11: $r = r''$
12: **end if**
13: **end for**

The star generation is an iterative process (Algorithm 2). First, the seed is extended against each negative example (line 3). The extension-against operator ($\dashv$) is a pair-wise generalization operation between the seed and a negative event aimed at finding the largest possible set of descriptions (rules) that cover the seed but not the negative. Thus, the result of the extension-against operation is a disjunction of single condition rules, namely one rule for each nonidentical attribute. An identical attribute is simply an attribute that has the same value for both the seed and the negative event, and for which a generalization that covers the seed but not the negative cannot be made. For each dimension, the largest possible description that covers the seed, but not the negative, is the negation of the negative. This definition of the extension-against operator only works for nominal attributes and is implemented by encoding the attribute domain into a binary vector, where each bit represented a particular value, and by negating this vector.

Assuming a nominal variable $||x|| = \{red, green, blue\}$, $x = blue$ is encoded as {0,0,1}. The result of the extension-against operation between a seed with $x =$ blue and a negative event with $x =$ red is the rule [positive] $\longleftarrow$ [$x =$ green or blue]. Its

binary representation is [{0,1,1}], which is exactly the negation of the negative event.

The extension-against operation for linear attributes is slightly different, and it involves flipping the bits only up to the value of the negative event, and not any values beyond. Assuming a linear variable $||y|| = \{XS, S, M, L, XL, XXL\}$, the result of the extension-against operation between a seed with $y = S$ and a negative event with $y = L$ is the rule [positive] $\longleftarrow$ $[y = XS \ldots M]$. Its binary representation is [{1,1,1,0,0,0}].

For integer and continuous attributes, the extension-against operator finds a value between the seed and negative. The degree of generalization can be controlled, and by default set to choose the middle point between the two. The ε parameter, defined between [0 and 1], controls the degree of generalization during the extension-against operation, with 0 being most restrictive to the seed, and 1 generalizing up to the negative. Assuming a continuous variable $||z|| = \{0.100\}$, the result of the extension-against operation between a seed with $z = 10$ and a negative event with $y = 30$, and ε = 0.5 is the rule [positive] $\longleftarrow$ $[z \leq 20]$. The result of the same extension-against operation with ε = 1 is [positive] $\longleftarrow$ $[z < 30]$ (note that 30 is not included).

The rules from the extension-against operation are then logically multiplied out with all the rules r to form a star (Algorithm 2, line 4), and the best rule (or rules) according to a given multicriterion functional LEF (Section *Lexicographical Evaluation Functions*) are selected (line 5). The parameter *maxstar* is central to the star generation process and defines how many rules are kept for each star.

If AQ is run in TF mode, the result from the intersection of the previously learned rules and the new rule is kept. In PD mode, the function Q [Eq. (5)] is used to compute the tradeoff between the completeness and the consistency of the rules

$$Q = \left(\frac{p}{P}\right)^{w} \left[\left(\frac{P+N}{N}\right)\left(\frac{p}{p+n} - \frac{P}{P+N}\right)\right]^{1-w} \quad (5)$$

where is p and n are the number of positive and negative events, and P and N are the total number of positive and negative events in the data. The parameter w is defined between 0 and 1 and controls the tradeoff between completeness and consistency.[9]

Lexicographical Evaluation Functions

LEF is an evaluation function composed from elementary criteria and their tolerances and are used to determine which rules best reflect the needs of the problem at hand. In other words, LEF is used to determine which rules, among those generated, are best suited to be included in the answer. AQ has been described as performing a *beam search* in space.[8] LEF is the parameter that controls the width of the beam.

LEF works as following:

1. Sort the rules in the star according to LEF, from the best to the worst.
2. Select the first rule and compute the number of examples it covers. Select the next rule and compute the number of new examples it covers.
3. If the number of new examples covered exceeds a new example threshold, then the rule is selected, otherwise it is discarded. Continue the process until all rules are inspected.

The result of this procedure is a set of rules selected from a star. The list of positive events to cover is updated by deleting all those events that are covered by these rules.

Postprocessing

Postprocessing operations consist in: (1) improvement of the learned rules through generalization and specialization operators, (2) optional generation of alternative covers, and (3) formatting of the output for textual and graphical visualization.

Optimization of Rules

When AQ is run in PD mode, rules can be further optimized during post processing. Rules can be generalized by dropping values in the reference of the conditions or by enlarging the ranges for continuous and integer attributes. Rules can be further generalized by dropping conditions altogether. Finally, entire rules could be dropped. The opposite operation of specialization is performed only at the condition level, by adding values in discrete attributes, and shrinking domains for integer and continuous attributes.

The optimization operation follows heuristics, and at each step computes what is called in AQ the Q value for the new rule Eq. (5). If the Q value increases, then the modified rule is added to the final answer, otherwise is disregarded.

Alternative Covers

Some of the rules learned during the star generation process, especially with large maxstar values, might not be required in the final output. The final step of the learning process consists in selecting from the pool of learned rules, only the minimum set required to

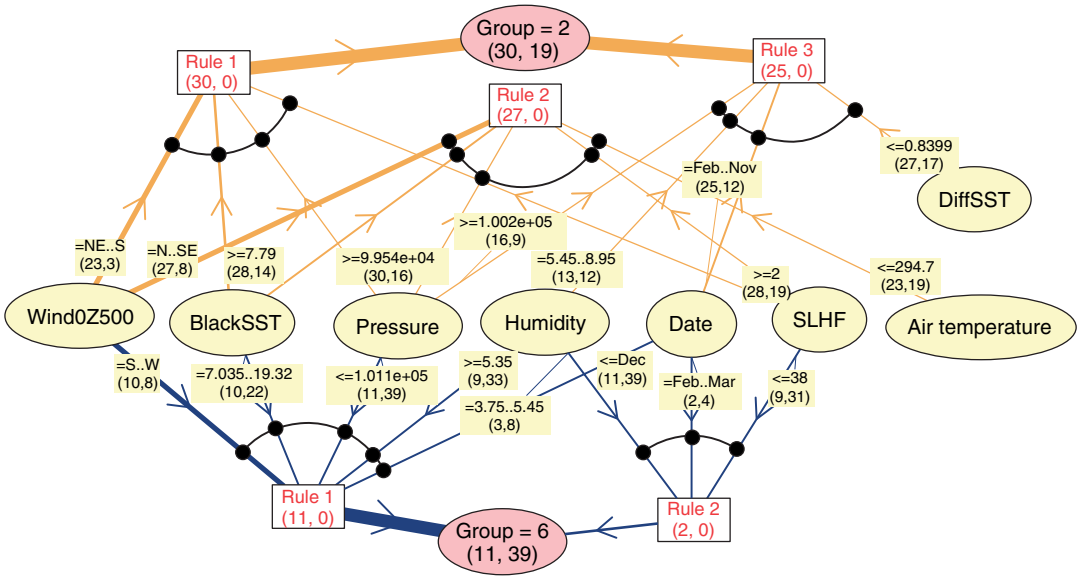


FIGURE 1 | A sample association graph from an atmospheric pollution problem.

cover the positive examples. Thus, some of the rules might not be included in the final answer and can be used to generate alternative solutions. Depending on the presence of multiple strong patterns in the data, alternative covers might be very useful to discriminate between classes.

Association Graphs

Association graphs are used to visualize attributional rules that characterize multivariate dependencies between target classes and input attributes. A program called concept association grapth (CAG) was developed by the first author to automatically display such graphs. Figure 1 is a graphical illustration of the rules discovered from an atmospheric release problem.[17] Representing relationships with nodes and links is not new nor unique to AQ and has been used in many applications in statistics and mathematics. Each target class is associated only with unique patterns of input parameters. The thickness of the links indicates the weight of a particular parameter-value combination in the definition of the cluster.

ADVANTAGES AND DISADVANTAGES OF THE AQ METHODOLOGY

The AQ methodology has intrinsic advantages and disadvantages with respect to other machine learning classifiers, such as neural networks, decision trees, or decision rules. Some of the original disadvantages have been solved or improved with additional components or optimization processes, often at the expense of a much slower or complex program. Other issues remain unresolved and open to investigation. The following discussion summarizes those that are believed to be the main issues to consider when choosing the AQ methodology over other methods, in particular C4.5 which is the closest widely used machine learning symbolic classifier.

Rich Representation Language

One of the main advantages of AQ consists in the ability to generate compact descriptions which are easy to read and understand. Unlike neural networks, which are black boxes and use a representation language that cannot be easily visualized, AQ rules can be inspected and validated by human experts. Although decision tree classifiers, such as C4.5, can convert the learned trees into rules, the resulting descriptions are expressed in a much simpler representation language in AQ. For example, C4.5 rules only allow for atomic relationships between attributes and possible values and do not allow for internal disjunctions or multiple ranges. Figure 2 shows the respective covers generated by AQ (left) and C4.5 (right). In this example, internal disjunction allows for a simpler and more compact representation due to intersecting patterns.

The cover generated by AQ [Eq. (6)] is composed of two rules with a single condition, each covering 20 positives and no negatives.

$$\begin{aligned}[\text{Positives} = 1] &\longleftarrow [X \geq 5] : p = 20, n = 0 \\ &\longleftarrow [Y \geq 5] : p = 20, n = 0 \quad (6)\end{aligned}$$

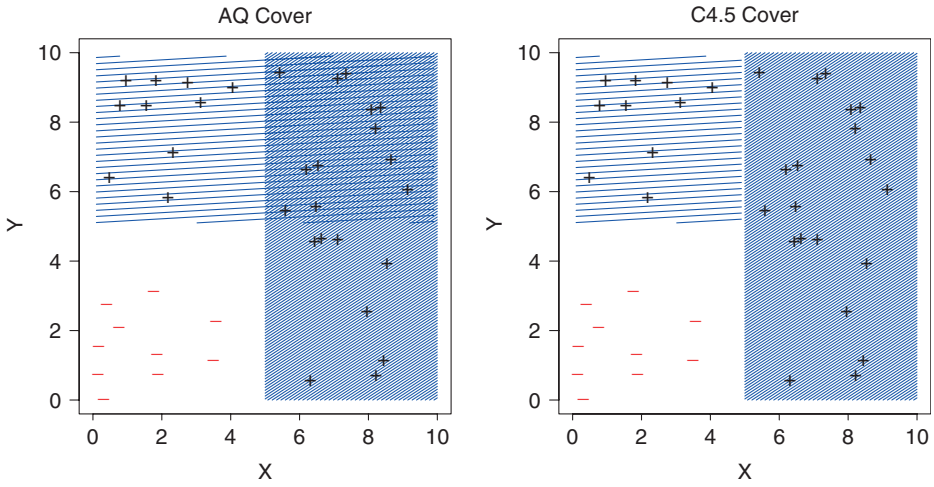
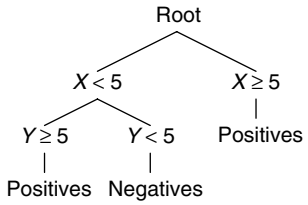


FIGURE 2 | Different covers generated by AQ (left) and C4.5 (right) using the same dataset.

In contrast, the tree [Eq. (7)] and the corresponding rules [Eq. (8)] generated by C4.5 cannot represent the intersecting concept because of the simpler representation language.



(7)

$$[\text{Positives} = 1] \longleftarrow [X \geq 5] : p = 20, n = 0$$
$$\longleftarrow [X < 5][Y \geq 5] : p = 10, n = 0 \quad (8)$$

The C4.5 cover is composed of two rules, one with a single condition, and one with two conditions. The first, identical to the rule learned by AQ, covers 20 positives and no negatives, whereas the second covers only 10 positives and no negatives. Although both covers are complete and consistent, the cover of C4.5 is more complex and cannot represent the intersecting concept.

Speed

AQ is considerably slower than C4.5 because of the underlying differences between the 'separate and conquer' learning strategy of AQ and the 'divide and conquer' strategy of C4.5. In C4.5, at each iteration, the algorithm recursively divides the search space. This means that at each iteration the algorithm analyzes an always smaller number of events. In contrast, AQ compares each positive with all of the negatives. Effectively, AQ can be optimized to consider only a portion of the positive examples but still has to consider all the negatives. This is in part due to the ability of representing intersecting concepts, meaning that rules are not bound to prior partitions.

Quality of Decisions

As previously seen, C4.5 performs consecutive splits on a decreasing number of positive and negative examples. This means that at each iteration, decisions are made on a smaller amount of information. In contrast, AQ considers all the search space at each iteration, meaning that all decisions are made with the maximum amount of information available.

Control Parameters

AQ has a very large number of control variables. Such controls allow for a very fine tuning of the algorithm, which can lead to very high quality descriptions. On the other hand, it is often difficult to determine *a priori* which set of parameters will generate better rules. Although heuristics on how to set the parameters exist, they are often suboptimal, and user fine tuning is required for optimal descriptions.

Matching of Rules and Events

AQ allows for different methods to test events on a set of learned rules. In C4.5, and most other classifiers which do not allow for intersecting concepts, testing an event usually involves checking if it is included in a rule or not. This is due to the fact the entire search space is partitioned into one of the target classes. In AQ, each event can be included in more than one rule or could be in an area of the search space which has not been assigned to any classes. Assigning an unclassified event to one of the target classes involves computing

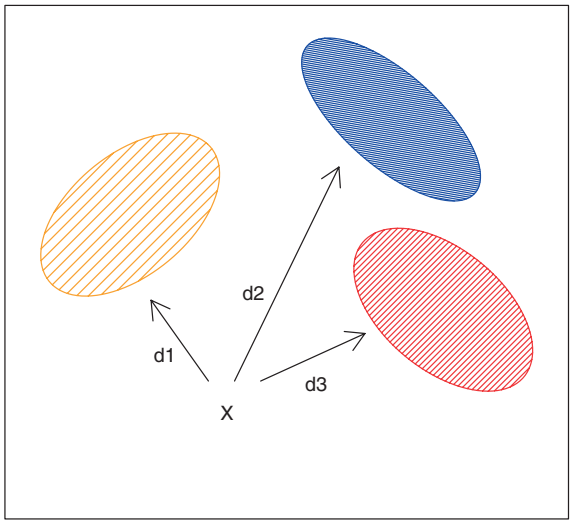


FIGURE 3 | An event that lies in an area of the search space which is not generalized to any of the training classes is assigned to the class it is closest to.

different degrees of match between the event and the covers of each of the classes and selecting the class with highest degree of match. Figure 3 shows the example of an unclassified event that lies in an area of the search space not generalized to any of the classes. The degree of match between the event and the cover of each of the class is computed, and it is assigned to the class with the highest score, in this case (d1). Several distance functions can be used to match rules and events and differ at the top level if they are strict or flexible.

In strict matching, AQ counts how many times a particular event is covered by the rules of each of the classes. An event can be covered multiple times by the rules of a particular class because the rules might be intersecting due to internal disjunctions. It can also be covered multiple times by rules of different classes if AQ was run in PD mode, and inconsistent covers were generated.

In flexible matching, AQ computes the ratio of how many of the attributes are covered over the total number of attributes. Assuming an event with three attributes, if a rule for class A matches three of them, and rule for class B matches two of them, the event is classified as type A because of a higher flexible degree of match. If the degree of match falls below a certain threshold, AQ classifies the event as unknown. In case more than one class has the same degree of match, the classification is uncertain, and multiple classes are output.

Multiple Target Classes

Decision tree classifiers are advantaged when learning from data with several target classes. They can learn descriptions for each of the classes using a single iteration of the algorithm, leading to very fast results. AQ, on the other hand, must be run multiple times, each time using the events of the target class as positives and the events of all the other classes as negatives. Such limitation seriously affects the execution time. Additionally, because rules are learned separately for each class, the resulting covers might be intersecting. Intersecting concepts might lead unseen testing events to be classified as belonging to more than one class.

Alternative Covers

AQ can generate alternative covers for each run. This is because in the postprocessing phase, only a portion of the learned rules are used for the final output. By selecting different rules combinations, it is possible to generate a number of alternative covers. Each cover might differ in completeness and consistency, and in simplicity of patterns.

Incremental Learning

Decision rule learners have the intrinsic advantage of being able to refine previous rules as new training data become available. This is because of the sequential nature of the 'separate and conquer' strategy of algorithms. Refinement of rules involves adding or dropping conditions in previously learned rules or splitting a rule in a number of partially subsumed rules. The main advantage is that modification in a rule of the cover does not affect the coverage of the other rules in the cover (Although the overall completeness and consistency of the entire cover might be affected). In contrast, although possible, it is more complicated to update a tree as it often involves several updates that propagate from leaves of the tree, all the way to the root. Additionally, the resulting tree might be suboptimal and very unbalanced, prompting for a complete re-evaluation of each node.

Input Background Knowledge

Because of the ability of AQ to update previously learned rules, it is possible to add background knowledge in the form of input rules. This feature is particularly important when there is an existing knowledge of the data, or constraints on the attributes, which can lead to a simpler rules and a faster execution.

EVOLUTIONARY COMPUTATION GUIDED BY AQ

The term evolutionary computation was coined in 1991 as an effort to combine the different approaches

to simulating evolution to solve computational problems.[18–23] Evolutionary computation algorithms are stochastic methods that evolve in parallel a set of potential solutions through a trial and error process. Potential solutions are encoded as vectors of values and evaluated according to an objective function (often called fitness function). The evolutionary process consists of selecting one or more candidate solutions whose vector values are modified to maximize (or minimize) the objective function. If the newly created solutions better optimize the objective function, they are inserted into the next generation, otherwise they are disregarded. While the methodologies and algorithms that are subsumed by this name are numerous, most of them share one fundamental characteristic. They use nondeterministic operators such as mutation and recombination as the main engine of the evolutionary process.

These operators are semi-blind, and the evolution is not guided by knowledge learned in the past generations, but it is a form of search process executed in parallel. In fact, most evolutionary computation algorithms are inspired by the principles of Darwinian evolution, defined by '. . .one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die'.[24] The Darwinian evolution model is simple and fast to simulate, and it is domain independent. Because of these features, evolutionary algorithms have been applied to a wide range of optimization problems.[25]

There have been several attempts to extend the traditional Darwinian operators with statistical and machine learning approaches that use history information from the evolution to guide the search process. The main challenges are to avoid local maxima and increase the rate of convergence. The majority of such methods use some form of memory and/or learning to direct the evolution toward particular directions thought more promising.[26–31]

Because evolutionary computation algorithms evolve a number of individuals in parallel, it is possible to learn from the 'experience' of entire populations. There is not a similar type of biological evolution because in nature there is not a mechanism to evolve entire species. Estimation of distribution algorithms (EDA) are a form of evolutionary algorithms where an entire population may be approximated with a probability distribution.[32] New candidate solutions are not chosen at random but using statistical information from the sampling distribution. The aim is to avoid premature convergence and to provide a more compact representation.

Discriminating between best and worst performing individuals could provide additional information on how to guide the evolutionary process. The learnable evolution model (LEM) methodology was proposed in which a machine learning rule induction algorithm was used to learn attributional rules that discriminate between best and worst performing candidate solutions.[33–35] New individuals were then generated according to inductive hypotheses discovered by the machine learning program. The individuals are thus genetically engineered, in the sense that the values of the variables are not randomly or semi-randomly assigned but set according to the rules discovered by the machine learning program.

The basic algorithm of LEM works like Darwinian-type evolutionary methods, that is, executes repetitively three main steps:

1. Create a population of individuals (randomly or by selecting them from a set of candidates using some selection method).
2. Apply operators of mutation and/or recombination to selected individuals to create new individuals.
3. Use a fitness function to evaluate the new individuals.
4. Select the individuals which survive into the next generations.

The main difference with Darwinian-type evolutionary algorithms is in the way it generates new individuals. In contrast to Darwinian operators of mutation and/or recombination, AQ conducts a reasoning process in the creation of new individuals. Specifically, at each step (or selected steps) of evolution, a machine learning method generates hypotheses characterizing differences between high-performing and low-performing individuals. These hypotheses are then instantiated in various ways to generate new individuals. The search conducted by LEM for a global solution can be viewed as a progressive partitioning of the search space.

Each time the machine learning program is applied, it generates hypotheses indicating the areas in the search space that are likely to contain high-performing individuals. New individuals are selected from these areas and then classified as belonging to a high-performance and a low-performance group, depending on their fitness value. These groups are then differentiated by a machine learning program, yielding a new hypothesis as to the likely location of the global solution.

To understand the advantage of using AQ to generate new individuals, compared with using the traditional Darwinian operation, it is necessary to take into account both the evolution length, defined as the number of function evaluations needed to determine the target solution, and the evolution time, defined as the execution time required to achieve this solution. The reason for measuring both characteristics is that choosing between the AQ and Darwinian algorithms involves assessing tradeoffs between the complexity of the population generating operators and the evolution length. The AQ operations of hypothesis generation and instantiation used are more computationally costly than operators of mutation and/or crossover, but the evolution length is typically much shorter than that of Darwinian evolutionary algorithms.

Therefore, the use of AQ as engine of evolution is only advantageous for problems with high objective function evaluation complexity. The problem of source detection of atmospheric pollutants described in this article is an ideal such problem because of the complexity of the function evaluation which requires running complex numerical simulations.

SOURCE DETECTION OF ATMOSPHERIC RELEASES

When an airborne toxic contaminant is released in the atmosphere, it is rapidly transported by the wind and dispersed by atmospheric turbulence. Contaminant clouds can travel distances of the order of thousand of kilometers within a few days and spread over areas of the order of thousands of square kilometers. A large population can be affected with serious and long-term consequences depending on the nature of the hazardous material released. Potential atmospheric hazards include toxic industrial chemical spills, forest fires, intentional or accidental releases of chemical and biological agents, nuclear power plants accidents, and release of radiological material. Risk assessment of contamination from a known source can be computed by performing multiple forward numerical simulations for different meteorological conditions and by analyzing the simulated contaminant clouds with clustering and classification algorithms to identify the areas with highest risk.[17]

However, often the source is unknown, and it must be identified from limited concentration measurements observed on the ground. The likely occurrence of a hazard release must be inferred from the anomalous levels of contaminant concentration measured by sensors on the ground or by satellite-borne remote sensors.

There are currently no established methodologies for the satisfactory solution to the problem of detecting the sources of atmospheric releases, and there is a great degree of uncertainty with respect to the effectiveness and applicability of existing techniques. One line of research focuses on the adjoint transport modeling,[36–38] but more general and powerful methodologies are based on Bayesian inference coupled with stochastic sampling.[39] Bayesian methods aim at an efficient ensemble run of forward simulations, where statistical comparisons with observed data are used to improve the estimates of the unknown source location.[40] This method is general, as it is independent of the type of model used and the type and amount of data, and can be applied to nonlinear processes as well. Senocak et al.[41] used a Bayesian inference methodology to reconstruct atmospheric contaminant dispersion. They pair the Bayesian paradigm with Markov chain Monte Carlo (MCMC) to iteratively identify potential candidate sources. A reflected Gaussian plume model is run for each candidate source, and the resulting concentrations are compared with ground observations. The goal of the algorithm is to minimize the error between the simulated and the measured concentrations.

A similar approach was followed by Refs 42–45, which use an iterative process based on genetic algorithms to find the characteristics of unknown sources. They perform multiple forward simulations from tentative source locations, and use the comparison of simulated concentration with sensor measurements to implement an iterative process that converges to the real source. The strength of the approach relies in the domain independence of the genetic algorithm, which can effectively be used with different error functions without major modifications to the underline methodology. The error functions quantify the difference between simulated and observed values.

The methodology applied in this article is based on this approach, but rather than using a traditional evolutionary algorithm, it uses AQ4SD to generate new individuals. This application is particularly suited for AQ4SD, because the function evaluation is very computationally intensive and requires running a numerical simulation. The main advantage of using AQ4SD to generate new individuals is the reduced number of function evaluations, which in this case translates to a huge improvement in speed.

Transport and Dispersion Simulations

Central to every evolutionary algorithm is the definition of the objective or fitness function. Given a candidate solution, the fitness function evaluates it and gives as feedback which solution is better for the

problem at hand. Each candidate solution is comprised of eight variables x, y, z, θ, U, Q, S, and ψ. x, y, and z are the coordinates of the release in kilometers; θ and U are, respectively, the wind direction and speed in degrees and ms^{-1}; Q is the source strength in gs^{-1}; S is proportional to the area of the release in m^2; and ψ describes the atmospheric stability according to Pasquill's stability classes.[46,47] The fitness of each candidate solution is computed using a normalized mean square error (NMSQE) function between the observed concentrations and the simulated values:

$$\text{NMSQE} = \sqrt{\frac{\overline{(C_o - C_s)^2}}{\overline{C_o}^2}} \tag{9}$$

where C_o is each sensor's observed values, and C_s is the corresponding simulated value. The bar indicates an average over all the observations. The values for C_s are simulated using a three-dimensional (3D) Gaussian dispersion model, that is,

$$C_s = P_1 P_2 (P_3 + P_4) \tag{10}$$

where $P_1, \ldots, P_4$ are defined by

$$P_1 = \frac{Q}{2\pi U \sqrt{(S+\sigma_y^2)(S+\sigma_z^2)}} \tag{11}$$

$$P_2 = \exp\left[-\frac{(y-y_0)^2}{2(S+\sigma_y^2)}\right] \tag{12}$$

$$P_3 = \exp\left[-\frac{(z-z_0)^2}{2(S+\sigma_z^2)}\right] \tag{13}$$

$$P_4 = \exp\left[-\frac{(z+z_0)^2}{2(S+\sigma_z^2)}\right] \tag{14}$$

where $\sigma_x(x, x_0; \psi)$, $\sigma_y(x, x_0; \psi)$, $\sigma_z(x, x_0; \psi)$ are the dispersion coefficients, which were computed from the tabulated equations of Briggs,[48] and $S = \sigma 2_y(x_o, x_o, \psi) = \sigma 2_y(x_o, x_o, \psi)$.

The result of the simulation is the concentration field generated by the release along an arbitrary wind direction. In order to map each C_s with the corresponding C_o, the wind direction θ is taken into account by applying a rotation to the x, y, and z coordinates of each C_s points.

Prairie Grass Experiment

The current application uses real-world data from the prairie grass field experiment.[49] The experiment consisted of 68 consecutive releases of 10 min each from the same source. SO_2 was used as a trace gas, and measurements of concentrations were made at

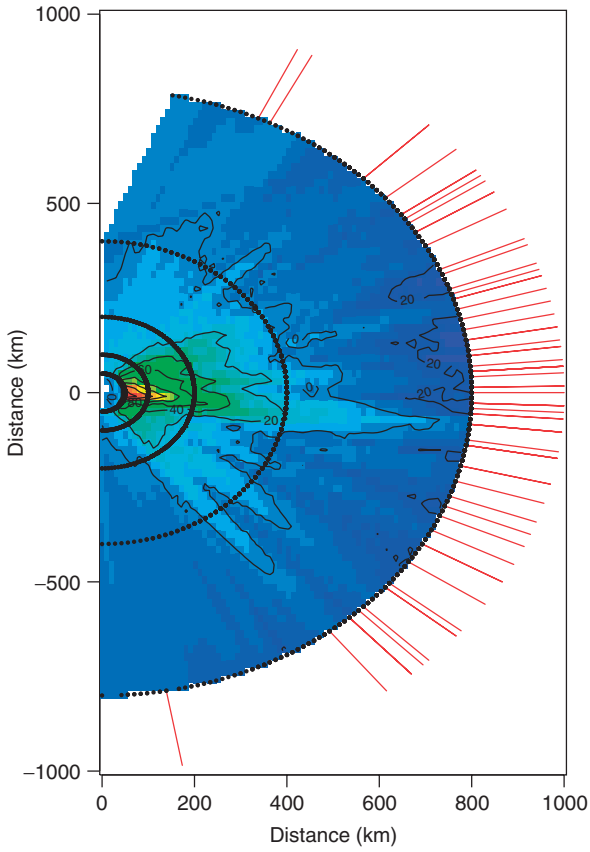


FIGURE 4 | Summary of the 68 prairie grass experiments.

sensors positioned along arcs radially located at distances of 50, 100, 200, 400, and 800 m from the source. Only sensors that recorded values above a minimum threshold were considered reliable, and as a result, each experiment has a different number of concentration measurements depending on the atmospheric conditions at the time of the release. The goal of the optimization process is to identify the source and the atmospheric characteristics. The only information used for the fitness evaluation are the values of the concentrations measured at the sensors. Figure 4 shows a summary of the 68 consecutive experiments. The concentration was computed by interpolating all the values measured at the concentric sensors (shown). The main direction of each release, as indicated in the experiment's summary, is shown with the solid lines protruding from the interpolated surface. One of the characteristics of the prairie grass experiment is the detailed information on the atmospheric conditions at the time of the release. It is then possible to classify each experiment as belonging to a different atmospheric type, using Pasquill's stability classes.[46,47] Pasquill's classes range from unstable (A) to neutral (D) to stably stratified atmosphere (F).

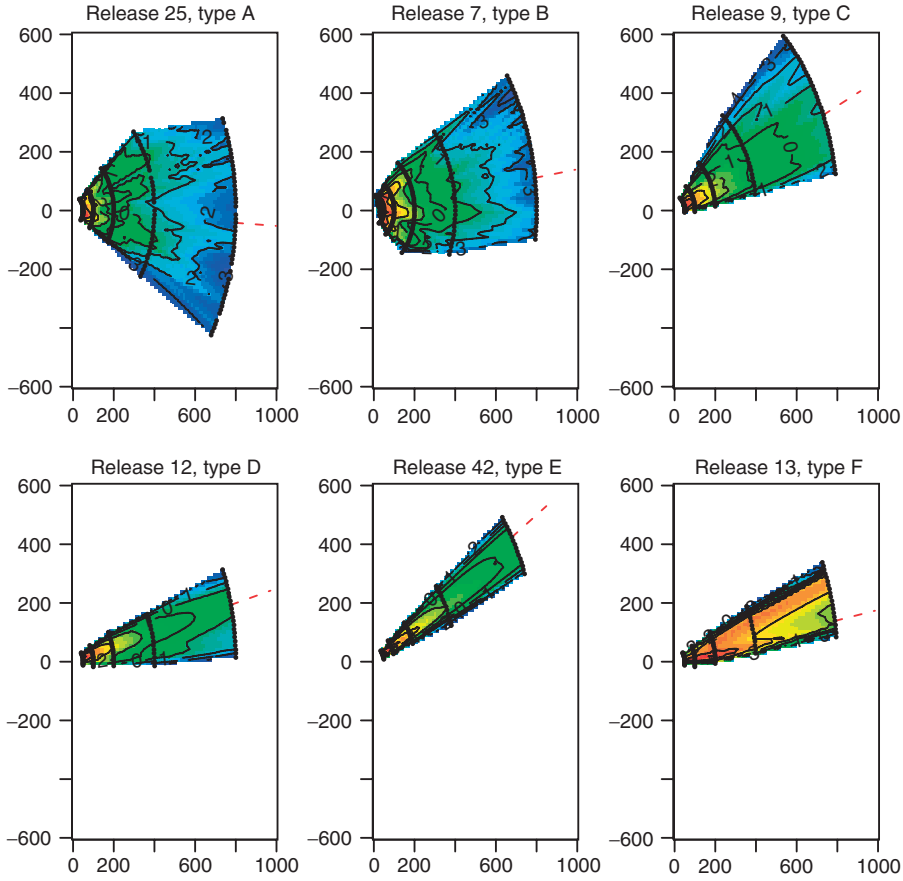


FIGURE 5 | Different sample prairie grass releases by atmosphere type.

Figure 5 shows 6 of the 68 experiments, each having occurred under a different atmospheric type. The figure shows how the atmospheric stability determines the characteristics of the concentration field. Unstable atmosphere (A) enhances the spread, thus reducing the ground level concentration, whereas stable atmosphere causes much narrower plumes, which result in higher ground concentrations.

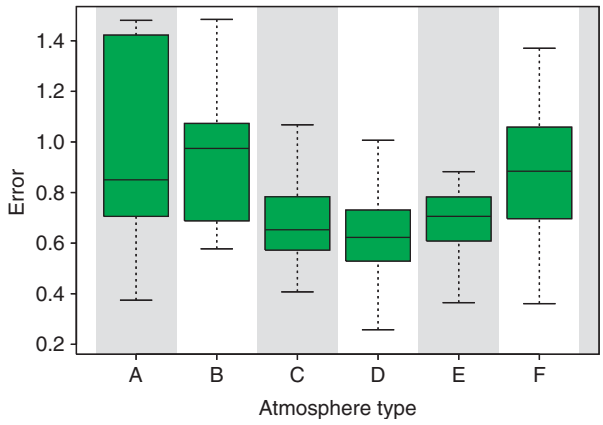


FIGURE 6 | Errors of AQ4SD divided by atmosphere type.

Results

Experiments were performed for each of the 68 prairie grass releases. The algorithm started by generating a population of random candidate solutions. Each candidate solution is a potential source and is encoded as a vector of eight variables: x, y, z, θ, U, Q, S, and ψ. For each potential source, the resulting concentration field is computed by Eq. (10). The fitness score of each source is defined as the error between the observed ground concentration and the simulated concentration at the same locations, computed according to Eq. (9).

The algorithm proceeds by dividing the candidate solutions with high and low fitness scores, and learning patterns (rules) which characterize the attribute values combinations that discriminate between the two groups. New candidate solutions are generated according to the learned patterns. The process continues for 500 iterations. The algorithm was run using a population of 100 candidate solutions. At each step, the top and lowest 30% of the solutions were used as members of the high- and low-performing groups. Each experiment was repeated 10 times to study the sensitivity of the

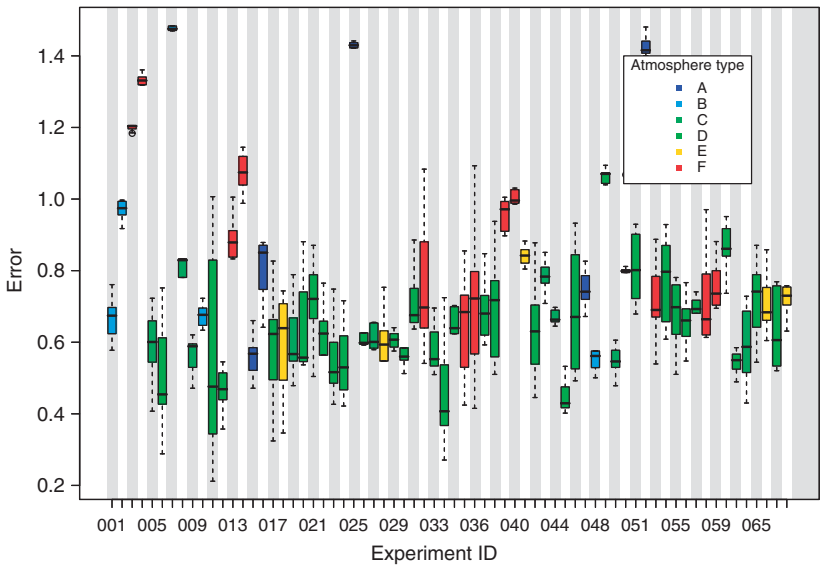


FIGURE 7 | Summary of the errors of AQ4SD for each prairie grass experiment. The atmosphere type of each experiment is color coded.

algorithm to the initial guess of solutions. The results are shown also in terms of atmospheric type.

A total of 680 AQ4SD source detections were performed, namely 10 for each of the 68 experiments. There is a considerably higher number of experiments of type D, as this was the predominant atmospheric condition at the time of the releases. In order to compensate for the different distributions of experiments, the results are normalized using this information.

Figure 6 shows a summary of the different errors, defined by Eq. (9), achieved by AQ4SD as a function of the atmospheric type. A threshold of 1.0 was assigned as minimum fitness value to recognize a source, because such value indicates AQ4SD identifying the source within 50 m of correct solution. Considerably better results were achieved for atmospheric type D, and worse results for atmospheric type A and type F. This pattern reflects the accuracy of the dispersion model (10) to reproduce the concentration field under different stability conditions. The Gaussian model is expected to perform better in neutral conditions (D), whereas convective turbulence (A) and stable stratification (F) involve more complex dispersion mechanisms which cannot be accounted for, resulting in a lack of accuracy. Figure 6 is consistent with the notion that the algorithm performs better when the fitness of the dispersion model is higher.

Figure 7 shows a summary for all the 68 prairie grass experiments. Each atmosphere type is color coded. With the exception of six experiments (3, 4, 7, 25, 52), each of type A or type F, AQ4SD always achieves a minimum fitness of 1.0, which was the target acceptance threshold for this experiment. The overall average fitness error is 0.6.

Figure 8 shows a summary of the results in terms of x, y, z, θ (called WA = wind angle), and Q. For all experiments, the original source was located at x,y,z = 0, 0, 0. The WA and Q errors are defined, respectively, as the identified angle minus the real angle, and the identified Q minus the real Q. Once again, the atmosphere type is color coded using the same colors as in Figure 7. The ideal solution would be all the points located at 0, for all variables. In the figure, although this is primarily true for most variables, the strong dependence between y and θ is evident. The units for the x,y, and z directions are meters. Therefore, the errors associated with changes in the z values are actually very small, as z only varies 10 m at most. There are larger errors for the alongwind dimension x compared with the crosswind dimension y. This is to be attributed to the concentration field which has a larger gradient in the crosswind direction compared with the alongwind direction. Note the correlation between the error in θ and the y dimension. Such behavior exemplifies the algorithm's skill at compensating for errors in y through changes in θ. The variable that seems to be harder to optimize is Q. Such results are primarily due to the correlation between Q and U [P1 in Eq. (10)].

DISCUSSION

This article introduces the main concepts of the AQ methodology and discusses its advantages and disadvantages. It describes a new implementation of the AQ methodology, AQ4SD, applied to the problem of source detection of atmospheric releases. In that context, AQ4SD is used as main engine of evolution

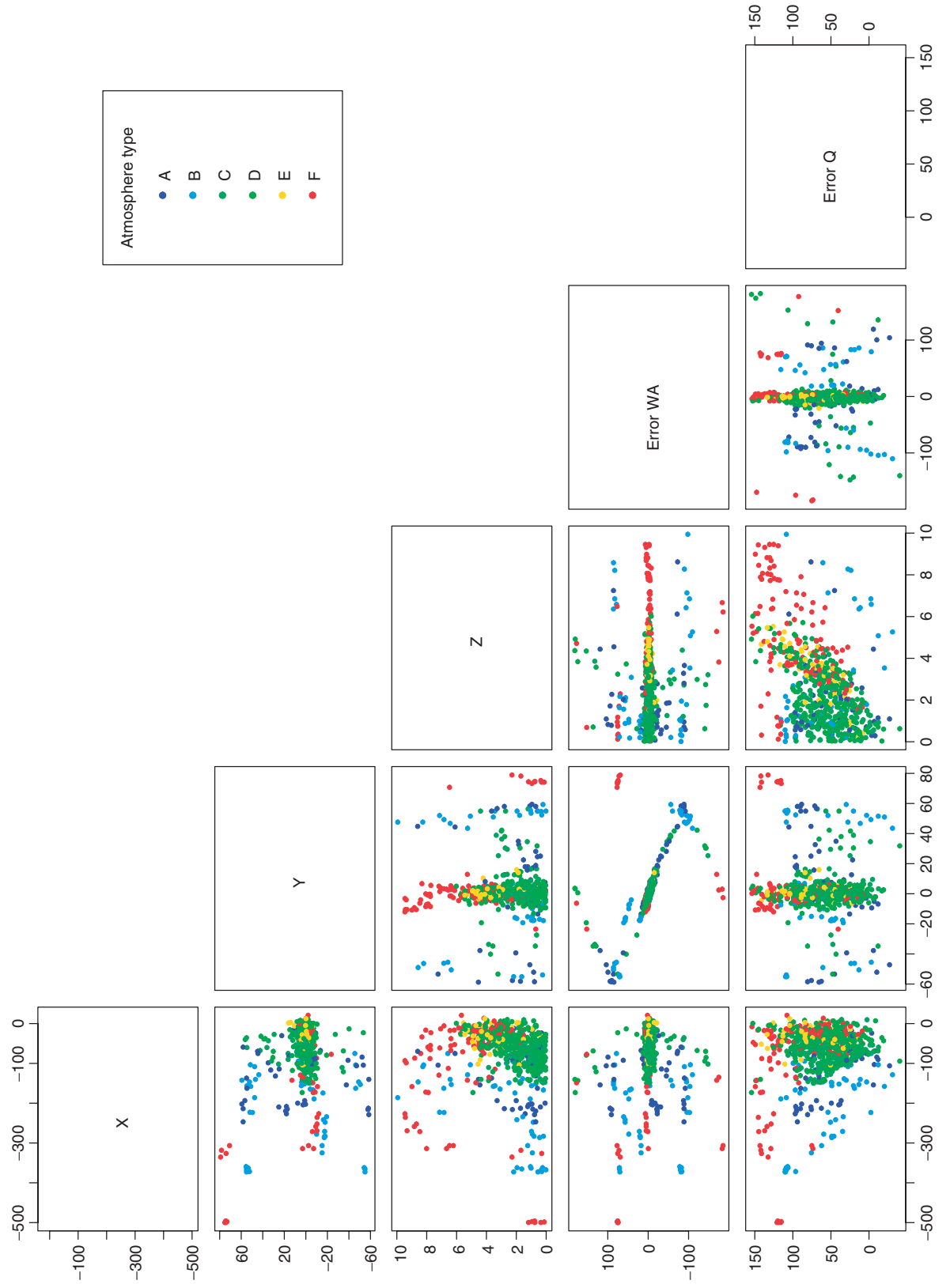


FIGURE 8 | Pair-wise plot of different attributes used during the optimization.

for an evolutionary computation process aimed at finding the source of an atmospheric release, using only a observed ground measurements and a numeric atmospheric dispersion model. Experiments were performed to identify the source of each of the 68 releases of the prairie grass field experiment.

The numerical experiments show that in all but five cases the methodology was able to achieve a fitness score considered acceptable for the correct identification of the source. The performance of the algorithm has been very satisfactory considering that the error intrinsic in the measured data and the approximation of the dispersion model. AQ4SD also proved to be quite efficient in terms of number of model simulations required for each optimization case. This is one of the main advantages of the proposed methodology compared with traditional evolutionary algorithms, because a fitness evaluation for a complete source detection procedure may require computationally expensive numerical simulations. In particular, for larger scale dispersion problems, more sophisticated and computationally expensive meteorological and dispersion models need to be run concurrently to evaluate the fitness of each candidate solution.[45]

The proposed methodology has a wide domain of applicability, not restricted only to the source detection problem. It can be used for a variety of optimization problems and is particularly advantageous for those problems where the fitness function evaluation involves a computationally expensive operation.

NOTES

[a]Some versions of AQ can also be run to generate rules with the largest amount of attributes (called characteristic mode), but such mode merely consists in generating discriminant rules and adding conditions that include all events in the class, but having no discriminatory information.

[b]Some versions of AQ sort all or a part of the negative events according to a distance metric. Although such mechanism has been shown to generate simpler rules in specific cases, because of the additional complexity of defining such distance metrics, which is not always possible as in the case of nominal attributes, paired with the additional computational resources required, the advantage of such sorting is not clear. AQ4SD can be run with and without sorting, and experiments have shown no or negligible improvements.

ACKNOWLEDGEMENTS

This material is partly based upon work supported by the National Science Foundation under Grant no: AGS 0849191.

REFERENCES

1. Michalski R. On the quasi-minimal solution of the general covering problem. *Proceesings of Fifth International Symposium on Information Processing (FCIP 69)*, Yugoslavia, Bled, vol A3; October 3–11 1969, 125–128.
2. Michalski R. A theory and methodology of inductive learning. *Mach Learn* 1983, 1:83–134.
3. Michalski R. *AQVAL/1 computer implementation of a variable-valued logic system VL 1 and examples of its application to pattern recognition. First International Joint Conference on Pattern Recognition*, Washington, D.C., 1973, 3–17.
4. Chilausky R, Jacobsen B, Michalski R. An application of variable-valued logic to inductive learning of plant disease diagnostic rules. *Proceedings of the Sixth International Symposium on Multiple-valued Logic*. Logan, UT: IEEE Computer Society Press Los Alamitos; 1976, 233–240.
5. Steinberg D, Colla P. *CART: Tree-structured Nonparametric Data Analysis* San Diego, CA: Salford Systems; 1995.
6. Quinlan J. *C4.5: Programs for Machine Learning*. San Mateo: Morgan Kaufmann; 1993.
7. Breiman L, Friedman J, Stone CJ, Olshen RA. *Classification and Regression Trees*: Wadsworth International Group; 1984.
8. Michalski R, Mozetic I, Hong J, Lavrac N. The multipurpose incremental learning system AQ15 and its testing application to three medical domains. *Proceedings of the 1986 AAAI Conference*, Philadelphia, PA vol. 104; August 11–15 1986, 1041–1045.
9. Kaufman K, Michalski R. *The AQ18 Machine Learning and Data Mining System: An Implementation and User's Guide. MLI Report*. Fairfax, VA: Machine Learning and Inference Laboratory, George Mason University; 1999.

10. Mitchell T. *Machine Learning*. New York: McGraw-Hill; 1997.
11. Cervone G, Panait L, Michalski R. The development of the AQ20 learning system and initial experiments. *Proceedings of the Fifth International Symposium on Intelligent Information Systems*, June 18-22, 2001, Zakopane, Poland: Physica Verlag; 2001, 13.
12. Keesee APK. *How Sequential-Cover Data Mining Programs Learn*. College of Science. Fairfax, VA: George Mason University; 2006.
13. Austern M. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. 1998.
14. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Westford, MA: Addison-Wesley Reading; 1995.
15. Bloedorn E, Wnek J, Michalski R. Multistrategy constructive induction: AQ17-MCI. *Rep Mach Learn Infer Lab* 1993, 1051:93–4.
16. Wnek J, Michalski R. Hypothesis-driven constructive induction in AQ17-HCI: a method and experiments. *Mach Learn* 1994, 14:139–168.
17. Cervone G, Franzese P, Ezber Y, Boybeyi Z. Risk assessment of atmospheric emissions using machine learning. *Nat Hazards Earth Syst Sci* 2008, 8:991–1000.
18. Holland J. *Adaptation in Natural and Artificial Systems*. Cambridge, MA: The MIT Press; 1975.
19. Goldberg DE. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley; 1989.
20. Bäck T. *Evolutionary Algorithms in Theory and Practice: Evolutionary Straegies, Evolutionary Programming, and Genetic Algorithms*. Oxford, NY: Oxford University Press; 1996.
21. Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed. Berlin: Springer-Verlag; 1996.
22. Fogel L. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming. Wiley Series on Intelligent Systems*. New York: John Wiley & Sons, Inc.; 1999.
23. De Jong K. Evolutionary computation: a unified approach. *Proceedings of the 2008 GECCO Conference on Genetic and Evolutionary Computation*. New York: ACM; 2008, 2245–2258.
24. Darwin C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: Oxford University Press; 1859.
25. Ashlock D. *Evolutionary Computation for Modeling and Optimization*. Berlin Heidelberg: Springer-Verlag; 2006.
26. Grefenstette J. Incorporating problem specific knowledge into genetic algorithms. *Genetic Alg Simul Annealing* 1987, 4:42–60.
27. Grefenstette J. *Lamarckian learning in multi-agent environments. Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1991.
28. Sebag M, Schoenauer M. *Controlling Crossover through Inductive Learning. Lecture Notes in Computer Science*. London: Springer-Verlag; 1994, 209–209.
29. Sebag M, Schoenauer M, Ravise C. *Inductive learning of mutation step-size in evolutionary parameter optimization, Lecture Notes in Computer Science*. London: Springer-Verlag; 1997, 247–261.
30. Reynolds R. *Cultural Algorithms: Theory and Applications. Mcgraw-Hill'S Advanced Topics In Computer Science Series*. Maidenhead, England: McGraw-Hill Ltd.; 1999, 367–378.
31. Hamda H, Jouve F, Lutton E, Schoenauer M, Sebag M. Compact unstructured representations for evolutionary design. *Appl Intell* 2002, 16:139–155.
32. Lozano J. *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*: Springer; 2006.
33. Michalski R. *Learnable evolution: combining symbolic and evolutionary learning. Proceedings of the Fourth International Workshop on Multistrategy Learning (MSL'98)*. 1999, 14–20.
34. Cervone G, Michalski R, Kaufman K, Panait L. Combining machine learning with evolutionary computation: Recent results on lem. *Proceedings of the Fifth International Workshop on Multistrategy Learning (MSL-2000)*. Portugal: Guimaraes; 2000, pp. 41–58.
35. Cervone G, Kaufman K, Michalski R. Experimental validations of the learnable evolution model. *Proceedings of the 2000 Congress on Evolutionary Computation*, LaJolla, CA, vol. 2; July 16–19 2000.
36. Pudykiewicz J. Application of adjoint tracer transport equations for evaluating source parameters. *Atmos Environ* 1998, 32:3039–3050.
37. Hourdin F, Issartel JP. Sub-surface nuclear tests monitoring through the ctbt xenon network. *Geophys Res Lett* 2000, 27:2245–2248.
38. Enting I. *Inverse Problems in Atmospheric Constituent Transport*. Cambridge, NY: Cambridge University Press; 2002, 392.
39. Gelman A, Carlin J, Stern H, Rubin D. *Bayesian Data Analysis*: Chapman & Hall/CRC; 2003, 668 pp.
40. Chow F, Kosović B, Chan T. *Source inversion for contaminant plume dispersion in urban environments using building-resolving simulations. Proceedings of the 86th American Meteorological Society Annual Meeting*, Atlanta, GA, January 2006, 12–22.

41. Senocak I, Hengartner N, Short M, Daniel W. Stochastic event reconstruction of atmospheric contaminant dispersion using Bayesian inference. *Atmos Environ* 2008, 42:7718–7727.

42. Haupt SE. A demonstration of coupled receptor/dispersion modeling with a genetic algorithm. *Atmos Environ* 2005, 39:7181–7189.

43. Haupt SE, Young GS, Allen CT. A genetic algorithm method to assimilate sensor data for a toxic contaminant release. *J Comput* 2007, 2:85–93.

44. Allen CT, Young GS, Haupt SE. Improving pollutant source characterization by better estimating wind direction with a genetic algorithm. *Atmos Environ* 2007, 41:2283–2289.

45. Delle Monache L, Lundquistand J, Kosović B, Johannesson G, Dyer K, et al. Bayesian inference and markov chain monte carlo sampling to reconstruct a contaminant source on a continental scale. *J Appl Meteor Climatol* 2008, 47:2600–2613.

46. Pasquill F. The estimation of the dispersion of windborne material. *Meteorol Magazine* 1961, 90:33–49.

47. Pasquill F, Smith F. *Atmospheric Diffusion*. Chichester, UK: Ellis Horwood; 1983.

48. Arya PS. *Air Pollution Meteorology and Dispersion*. Oxford, NY: Oxford University Press; 1999.

49. Barad M, Haugen D. *Project Prairie Grass, A Field Program in Diffusion*: United States Air Force, Air Research and Development Command, Air Force Cambridge Research Center; Cambridge, MA, 1958.

