Published in final edited form as:
Evol Intell. 2015 September ; 8(2): 89–116. doi:10.1007/s12065-015-0128-8.

ExSTraCS 2.0: Description and Evaluation of a Scalable Learning Classifier System

Ryan J. Urbanowicz and
Geisel School of Medicine, 1 Medical Center Dr., Lebanon NH, 03756, USA, Tel.: +603-653-6017, Fax: +603-653-9952

Jason H. Moore
Geisel School of Medicine, 1 Medical Center Dr., Lebanon NH, 03756, USA, Tel.: +603-653-6017, Fax: +603-653-9952

Ryan J. Urbanowicz: ryan.j.urbanowicz@dartmouth.edu; Jason H. Moore: jason.h.moore@dartmouth.edu

Abstract

Algorithmic scalability is a major concern for any machine learning strategy in this age of ‘big data’. A large number of potentially predictive attributes is emblematic of problems in bioinformatics, genetic epidemiology, and many other fields. Previously, ExS-TraCS was introduced as an extended Michigan-style supervised learning classifier system that combined a set of powerful heuristics to successfully tackle the challenges of classification, prediction, and knowledge discovery in complex, noisy, and heterogeneous problem domains. While Michigan-style learning classifier systems are powerful and flexible learners, they are not considered to be particularly scalable. For the first time, this paper presents a complete description of the ExS-TraCS algorithm and introduces an effective strategy to dramatically improve learning classifier system scalability. ExSTraCS 2.0 addresses scalability with (1) a rule specificity limit, (2) new approaches to expert knowledge guided covering and mutation mechanisms, and (3) the implementation and utilization of the TuRF algorithm for improving the quality of expert knowledge discovery in larger datasets. Performance over a complex spectrum of simulated genetic datasets demonstrated that these new mechanisms dramatically improve nearly every performance metric on datasets with 20 attributes and made it possible for ExSTraCS to reliably scale up to perform on related 200 and 2000-attribute datasets. ExSTraCS 2.0 was also able to reliably solve the 6, 11, 20, 37, 70, and 135 multiplexer problems, and did so in similar or fewer learning iterations than previously reported, with smaller finite training sets, and without using building blocks discovered from simpler multiplexer problems. Furthermore, ExS-TraCS usability was made simpler through the elimination of previously critical run parameters.

Keywords

Learning Classifier System; Scalability; Evolutionary Algorithm; Data Mining; Classification; Prediction

1 Introduction

Learning classifier systems (LCSs) are an often overlooked group of rule-based machine learning algorithms with a unique and flexible set of features setting them apart from other strategies. Two major genres of LCS algorithms exist, including Michigan-style and Pittsburgh-style systems [46]. Michigan-style LCS algorithms are the more traditional of the two LCS architectures and are the focus of the present work. Michigan LCS algorithms uniquely distribute learned patterns over a population of individually interpretable (IF: THEN) rules that are both collaborative and competitive. This allows the algorithm to flexibly and effectively describe complex and diverse problem spaces found in behavior modeling, function approximation, classification, and data mining. Michigan LCS algorithms also apply incremental rather than batch-wise learning, meaning that rules are evaluated and evolved one training instance at a time rather than being immediately evaluated on the training dataset as a whole. This makes them efficient, and naturally well-suited to learning different problem niches found in multi-class, latent-class, or heterogeneous problem domains [37].

The nature of Michigan LCSs impart them with a number of notable advantages: (1) they are model-free and thus do not make assumptions about the data including; the type of association (e.g. linear, epistatic, or heterogeneous); the number of predictive attributes; noisy or clean signal; balanced or imbalanced classes; how to handle missing data; or whether attributes are a mix of both discrete and continuous variables, (2) Individual rules are directly interpretable as IF:THEN expressions, (3) they are implicitly multi-objective, evolving rules toward maximal accuracy and generality (i.e. rule simplicity) to improve predictive performance [46], (4) on their own, they have themes in common with an ensemble learning system where individual rules or groups of rules propose different solutions within a collective rule population, (5) they are adaptive to changing dataset environments, such that parts of a solution can adapt without starting over from scratch, (6) by relying on evolutionary computation to explore the search space they offer a practical option when deterministic, exhaustive search is intractable, and (7) the algorithm is compartmentalized such that individual components can be updated or replaced to handle specialized problem domains. Key disadvantages of LCS include (1) the belief that they are somewhat more difficult to properly apply, (2) they lack a comparable theoretical understanding next to other, well-known machine learning strategies, (3) they are relatively computationally demanding and in certain problem domains can take longer to converge on a solution, and (4) most implementations to date have a relatively limited scalability.

Within the LCS literature, scalability and learning speed have been largely synonymous targets for improvement [2,27,6,1,17,26,5]. Much of the progress in dealing with scalability to date has been made in the development of Pittsburgh-style LCS architectures [2,6,1,17,5]. The term scalability can refer to an algorithms ability to handle different types of increased size in the training environment. This can include (1) an increasing number of training instances (of particular importance to Pittsburgh-style systems, where rule-sets are typically repeatedly evaluated on the training dataset as a whole), (2) an increase in the problem dimensionality, i.e. the number of attributes required to accurately predict an endpoint variable, and (3) an increasing number of total attributes within which the algorithm must search for the correct set of predictive attributes.

To date, strategies to improve LCS scalability have included parallelization of the algorithm [17,33], more efficient rule representations [1,3,24], more efficient matching [27], running the algorithm as an ensemble where individual runs have access to only a subset of available attributes in the dataset [5], extracting and using building blocks from smaller problems [23], applying expert knowledge to bias the algorithm towards attribute more likely to be of predictive value [43,41], and using windowing mechanisms in Pittsburgh LCSs to learn incrementally [2].

In this work we focus on the scalability of Michigan LCS algorithms on problems with an increased number of attributes and problem dimensionality. This is an accessible and important target for development in the context of the Michigan LCS, where the number of training instances in the dataset does not necessarily impact the run-time of the algorithm. Handling an increasing number of potentially predictive attributes can be viewed as a feature selection challenge where the algorithm must not only identify which attributes are important, but the manner in which they are associated with the target endpoint.

Two features of a traditional Michigan LCS intuitively complicate the ability of the algorithm to scale up to handle larger numbers of attributes. First, existing algorithms rely on the probability of attribute specification (p_{spec}) as a critical run parameter. This parameter controls the number of attributes that might be specified in a rule generated by the covering mechanism. The covering mechanism is responsible for initializing the rule population. As the number of attributes in the dataset becomes large, the optimal setting for this parameter becomes difficult or impractical to determine. The most critical consequence of it being set too high, is that the rule population will be repeatedly seeded with rules that specify too many attribute values and therefore are unlikely to match more than a single training instance. This means that useful generalizations will not be found. Beyond covering, the genetic algorithm provides the only other source of rule discovery. The second complication to scalability involves the nature of crossover and mutation operators within the LCS genetic algorithm. Specifically, these mechanisms typically allow the discovery of rules that specify any number of attributes in the dataset. While this is maximally unbiased in terms of limiting the dimensionality of rules, when we scale up to much larger datasets rules regularly emerge that are not only over-specific and over-fit, but that do not match more than a single training instance. Consider that in typical LCS rule representations, mutation can only flip an attribute from generalized to specified (and vice versa), or it can swap between 0,1, or '#' when the classic ternary representation is applied. Given the example where a rule with only 2 out of 100 attributes are specified, mutation will be much more likely to specify additional attributes in this rule, rather than generalize the two attributes already specified (i.e. 2/100 chance of further generalization, and 98/100 chance of further specification). Therefore an implicit pressure exists that mutates rules towards having half of all attributes in the dataset specified or two thirds specified in the case of a ternary representation by random chance. While this might not be problematic for small datasets, as the number of attributes increases, the probability that a rule will evolve that is general enough to cover more than one training instance will decrease.

To address these issues, the present study explores the use and implementation of a rule specificity limit that essentially limits the number of attributes that may be specified within a given LCS rule. This limit is automatically determined based on the characteristics of the training data before the algorithm begins learning. In this work we introduce both a strategy for determining a reasonable specificity limit and explore a strategy to efficiently utilize this limit within the mechanisms of the algorithm responsible for discovering new rules (i.e. covering, crossover, and mutation).

Previously, we developed an Extended Supervised Tracking and Classifying System (ExSTraCS) as a promising platform to address supervised learning problem domains dealing with classification, prediction, and/or knowledge discovery [41]. Specifically, we were interested in developing and applying this algorithm to the domain of bioinformatics and genetic epidemiology. A typical problem in this domain involves the identification and modeling of predictive single nucleotide polymorphisms (SNPs) that have a complex and noisy relationship with a disease endpoint of interest. Complexity in this domain refers to an unknown number of predictive attributes, the potential for epistasis (i.e. gene-gene interactions), and genetic heterogeneity, where individual (or sets of) attributes are independently predictive of the same disease phenotype (i.e. class). In the present study we assess the scalability of ExSTraCS 1.0 to accommodate an increasing number of potentially predictive attributes (20, 200, and 2000) across a diverse spectrum of complex simulated datasets. We also introduce and similarly evaluate ExSTraCS 2.0 which implements our proposed rule specificity limit, new approaches to expert knowledge guided covering and mutation mechanisms, and the addition of Tuned ReliefF (TuRF)[28] with one minor alteration. TuRF was designed to improve the quality of expert knowledge scores on data with a larger number of noisy attributes when combined with one of the four attribute scoring algorithms included in ExSTraCS. Beyond our complex genetic simulation studies we also evaluate ExSTraCS 2.0 on a spectrum of multiplexer datasets scaling up from the 6-bit to 135-bit version of this benchmark problem.

In the following sections we describe the ExSTraCS algorithm in detail, outline the experimental evaluation, present results and discussion, and draw conclusions.

2 Methods

In this section, we describe the ExSTraCS algorithm and it’s mechanisms in detail, highlighting features that are new to version 2.0. If you are new to LCS algorithms, this section will hopefully provide a clear and complete account of how this type of algorithm works. If you are familiar with LCS algorithms we refer readers below to specific subsections that may be of particular interest. For a brief overview of the ExSTraCS algorithm, see subsection 2.1.2. At the end of this section we have outlined our strategy for evaluating ExSTraCS performance and scalability including a description of simulated datasets, performance metrics, and statistical analysis.

The primary update to ExSTraCS 2.0 involves the rule specificity limit. Determination of this limit within adaptive data management is covered in subsection 2.1.4. The inclusion of the rule specificity limit has required us to adapt how expert knowledge is applied within the covering mechanism (see subsection 2.1.9). Application of this limit within genetic algorithm operators (mutation, and crossover) and the development of a new EK mutation scheme is covered in subsection 2.1.13. Also, to improve the quality of expert knowledge scores generated using the built-in attribute weighting algorithms (discussed later) we have added the TuRF wrapper algorithm [28] to ExSTraCS as part of the preprocessing expert knowledge discovery (see subsection 2.1.5). Lastly, we have added a new optional output file giving users the ability to view the predictions made by ExSTraCS on a testing data (see subsection 2.1.17). For each testing instance in the data this output includes the class prediction as well as the vote sum for each possible class. These vote sums may be utilized as fuzzy predictions, Bayesian class probabilities, or estimates of prediction confidence on given testing instances.

2.1 ExSTraCS Algorithm

The Extended Supervised Tracking and Classifying System (ExSTraCS) [41] was implemented as a platform for ongoing Michigan LCS development designed specifically to address single-step, supervised learning problems, dealing with data mining, classification, prediction, and knowledge discovery. In particular, development of this algorithmic platform is geared towards problems characterized by complex patterns of association (e.g. multi-factor epistasis and heterogeneity), the integration of different data types (e.g. systems biology), and 'big data' (i.e. large-scale data and analyses). ExSTraCS was implemented to accommodate datasets with 2 or more balanced/imbalanced endpoints (i.e. classes), with continuous or discrete attributes, and allowing for missing data points without requiring data imputation.

2.1.1 Background—The ExSTraCS algorithm [41] is descended from a lineage of Michigan LCS algorithms founded primarily on the architecture of XCS [48], the most well-known and best-studied LCS algorithm to date. The UCS algorithm [9] replaced XCS's reinforcement learning scheme with a supervised learning strategy to deal explicitly with single-step problems such as classification and data mining. Previously, we compared select Michigan and Pittsburgh-style LCS algorithms and found that UCS showed particular promise when applied to complex biomedical data mining problems with patterns of epistasis and heterogeneity [38,39]. UCS inspired a series of algorithmic extensions that better adapted the LCS algorithmic framework to the needs of complex noisy epidemiological data mining problems [37,43,35]. These extensions, along with a flexible and efficient rule representation [3] and a handful of other useful features, were merged within the ExSTraCS framework [41]. The combination of these features served to further improve performance over a range of metrics detailed in section 2.2.2.

2.1.2 Overview—Here we provide a brief overview of the ExSTraCS algorithm. The diagram in Figure 1 sequentially outlines the flow of ExSTraCS including a pre-processing stage (A), the core incremental learning steps of the algorithm (B), and a post-processing stage (C). It should be noted upfront that major features of this algorithm can be optionally turned on or off including the pre and post-processing mechanisms. In this schematic, ovals indicate algorithmic mechanisms, bordered squares indicate sets of either data or classifiers/ rules, items in green are rule discovery mechanisms, items in orange are traditional LCS mechanisms, items in blue indicate mechanisms that are unique to ExSTraCS, and items with a (*) indicate mechanisms that have been updated or added in ExSTraCS 2.0.

Following along with Figure 1, ExSTraCS begins with a data pre-processing stage (A). The algorithm takes as input, a configuration file and a finite training dataset that includes attributes and a single class endpoint variable. Adaptive data management determines and stores key characteristics of the dataset for later use. Next, expert knowledge (EK) attribute weights may be loaded or generated from the training data using one of the four included attribute weighting algorithms. Weights are utilized by the discovery mechanisms to favor the exploration of rules that specify attributes with higher weights. Attributes with higher weights should be more likely to be useful in discriminating between class/endpoints. After pre-processing, the core incremental learning cycle (B) repeats through the following 10 steps until a maximum number of learning iterations (I_{max}) is reached. (1) A single training instance is taken from the dataset without replacement. (2) The training instance is passed to a population [P] of classifiers that is initially empty. A *classifier* is a simple IF/THEN rule comprised of a condition (i.e. specified attribute states), and what is traditionally referred to as an action (i.e. the class, also referred to as the endpoint or phenotype). Note that herein, the terms *classifier* and *rule* are used interchangeably. (3) A match set [M] is formed, that includes any classifier in [P] that has a condition matching the training instance. (4) [M] is divided into a correct set [C] and an incorrect set [I] based on whether each classifier specified the correct or incorrect class phenotype. (5) If, after steps 3 and 4, [C] is empty, covering applies expert knowledge to intelligently generate a matching and ‘correct’ classifier that is added to [M] and [C]. (6) For every classifier in [P], a number of parameters are maintained and updated throughout the learning process. Classifier parameters (e.g. fitness) are updated for classifiers within [C] and [I]. (7) Subsumption provides a direct generalization pressure that can merge overlapping rules in [C]. A similar subsumption mechanism is also applied to new classifiers generated by the genetic algorithm (GA). (8) Classifiers in [C] are used to update attribute tracking scores for the current training instance. These scores can be fed back into the GA to bias discovery toward useful attribute combinations. (9) The GA uses tournament selection to pick two parent classifiers from [C] based on fitness (i.e. a niche GA) and generates two offspring classifiers which are added to [P]. The GA includes two discover operators; uniform crossover and mutation. (10) Lastly, whenever the size of [P] is greater than the specified maximum (N), a deletion mechanism selects a rule and either decrements the numerosity of the rule (if there are multiple copies), or removes it from [P] entirely (if it is the last copy). After all learning iterations have completed, rule compaction is applied as a post-processing step (C) to remove poor and/or redundant rules from [P] to yield [P_c]. After completion, the ExSTraCS output may be used to make class predictions or evaluated and visualized to facilitate knowledge discovery.

2.1.3 Input—See ‘Input’ in (A) of Figure 1. At minimum, ExSTraCS requires a configuration file and a training dataset to run. The configuration file is a formatted (.txt) file which specifies algorithm run parameters. A detailed overview of all run parameters and instructions for running the algorithm are included in the ExSTraCS Users Guide [36]. Two run parameters that are particularly important to successful performance include the maximum rule population size (N), and the maximum number of learning iterations (I_{max}). The training data constitutes the ‘environment’ within which ExSTraCS seeks to learn. The training data file should be a tab-delimited (.txt) file, where the first row includes column headers (i.e. identifiers for attributes, the class variable, and instance identifiers, if available). Any missing values in the dataset should have a standard unique designation.

2.1.4 Adaptive Data Management—See 'Adaptive Data Management' in (A) of Figure 1. The adaptive data management (ADM) component was originally designed to facilitate ease of use, improve learning efficiency, and automatically adapt ExSTraCS to learn on datasets that might include either discrete or continuous attributes [41]. ADM will load and format training (and optionally testing) data, simultaneously identifying key characteristics including: number of attributes, number of instances, the location of the endpoint variable column, and (optionally) the location of the instance identifiers column. Upon loading, the order of the training instances is randomized once in preparation for learning. This avoids any potential bias that might be introduced by the initial ordering of the data. ADM automatically identifies and stores which attribute variables to treat as discrete, and which to treat as continuous. It does this by counting the number of unique state values for each attribute, and checks if this value is greater than a user defined run parameter (*discreteAttributeLimit*). If continuous, the maximum and minimum values are stored, for use in limiting evolved attribute ranges within rules.

In this work we have added the calculation of a rule specificity limit to the ADM. The rule specificity limit (RSL) is simply the maximum number of attributes that may be specified within a given rule. Essentially this limits the dimensionality of a given rule, or the maximum order of interaction that it can specify. In the context of our genetic epidemiological problem of interest, an RSL of 7 means that the ExSTraCS algorithm has the flexibility to search for up to a 7-way epistatic interaction. RSL removes some of the algorithm's flexiblity, in that it adds the assumption that the underlying pattern of association does not require more than, for example, 7 unique attribute states to be specified together in order to make an accurate prediction. Importantly, the RSL does not limit the number of total attributes that can be involved in the overall prediction model. For instance, even with RSL of 5, ExSTraCS can easily solve the 6-bit multiplexer problem, wherein 6 attributes are critical to the overall solution, but optimal rules only specify 3 attributes. Keep in mind that with LCS algorithms, the entire rule population is collectively the prediction 'model'.

Rather than pick an arbitrary value for RSL we have implemented a simple, naive approach to automatically set the RSL to a reasonable, functional value. This strategy is based on the concept of power, where we want to estimate whether we have a sufficient number of training instances to detect an interaction of order *n* (i.e. an *n*-way interaction). We pick a value for RSL such that we prevent the search for *n*-way interactions that we have little to no hope of detecting, given the number of training instances (ι) available in our dataset. We estimate this cutoff by comparing ι to the number of unique attribute state combinations (ψ) for a given *n*-way interaction.

$$\psi=\varepsilon^{n} \quad (1)$$

The value ψ is calculated using the average number of unique discrete states (ε) across attributes in the training data. For reference, the multiplexer problem has an ε of 2 (every attribute can either have a value of 0 or 1), and the SNP data explored in this paper has an ε of 3 (every attribute can have one of three genotypic states; 0, 1, or 2). For simplicity, if an attribute is continuous-valued, we count the variable as having two states when calculating ε. Notably, this study only explores discrete-valued datasets, therefore another approach to calculating ε in the presence of continuous attributes may improve performance.

RSL is selected by sequentially increasing n until $\iota < \psi$. When this occurs, RSL= n. This comparison between ι and ψ essentially seeks to determine when the 'curse of dimensionality' [31] might spread training instances so thinly over the interaction model, that if we assume the data is uniformly distributed over all combinations, we would have no more than one training instance per attribute combination. If this were the case, we would have not have sufficient power to identify that model to be predictive. While it is very unlikely that the training data would be uniformly distributed in this way, this comparison provides a convenient, liberal, and data-driven strategy to pick the RSL. We do not claim that this strategy for selecting RSL is optimal, but instead we seek to demonstrate that the use of a reasonable RSL improves performance and scalability. Table 1 gives calculations of ψ for different combinations of n and ε. Using this table, we can see that for SNP datasets (with an ε of 3 and $\iota = 1440$) ExS-TraCS will use a RSL of 7, or in other words it will be limited to searching for up to a 7-way pure, strict epistatic interaction between SNP attributes [45]. For comparison, consider that very few epidemiological investigations ever consider interactions of an order larger than 3 or 4 [30] due to the outrageous computational demands of traditional exhaustive methodologies.

RSL replaces the need for the p_{spec} parameter which has been used in most Michigan LCS algorithms to determine the probability that individual attributes might be specified during covering. Additionally, p_{spec} was used in ExSTraCS 1.0 to transform expert knowledge scores into probabilities if expert knowledge covering was activated [41]. The implementation of RSL also eliminates the need for two additional parameters (EK_{max}, and EK_{dop}), previously required for the logistic transformation of expert knowledge scores into probabilities that could be used by the algorithm [43, 41]. For user flexibility, we have added an optional parameter to ExSTraCS 2.0 that will override the automated RSL calculation, allowing the user to directly specify their own RSL. Overall, the ADM improves efficiency, makes ExSTraCS more user friendly, and paves the way for future algorithmic enhancements.

2.1.5 Expert Knowledge Discovery—See 'Expert Knowledge Discovery' in (A) of Figure 1. Expert knowledge (EK) is essentially an external bias introduced to the algorithm that encourages learning to more frequently explore attributes that have a better chance to be useful in predicting the class endpoint of interest. When EK is applied, rules will tend to be generated that cover parts of the search space believed to be of greater importance. Notably, the utility of EK is only as good as the quality of the information driving the weights.

Previous work explored the utilization of EK within the UCS algorithm [43]. The results indicated that EK, utilized as probabilistic weights for specifying attributes in rules, significantly sped up learning when applied to covering, but yielded inconsistent success when applied to the mutation operator. Specifically, while EK mutation significantly improved key metrics like testing accuracy and the power to find underlying predictive attributes, it harmed the ability to fully characterize underlying heterogeneity. This was attributed to the global pressure of EK scores (always active in the genetic algorithm) adding consistent bias towards specifying attributes with the highest EK scores, but a lesser bias towards predictive attributes with a weaker signal. ExSTraCS 1.0 incorporated the EK covering strategy as described in that earlier work. In order to effectively scale up ExSTraCS 2.0 to handle larger numbers of attributes, we have introduced a new strategy to incorporate expert knowledge into both covering and the mutation operator described further in sections 2.1.9 and 2.1.13.

The source of EK is ultimately up to the user. ExS-TraCS can load a formatted file including custom EK scores for each attribute in the dataset, or generate EK scores directly from the training data using one of four Relief-based attribute weighting algorithms implemented into ExSTraCS. These include ReliefF [25], SURF [19], SURF* [20], and MultiSURF [18]. Each of these algorithms have been coded to allow for both discrete and continuous attributes. Relief-based algorithms are based on the assumption that by comparing training instances that are maximally similar (neighbors), we can derive potentially predictive information by focusing on attributes in these neighboring rules that have different states (e.g. different genotypes). Figure 2 illustrates this core concept. In this example we have a pair of training instances that are considered to be neighbors since their attributes states are similar (i.e. all but two attributes in this example have identical states (either A, B, or C). In this example the two instances belong to different classes (0 and 1), referred to as a ‘miss’. A miss between neighboring instances, means that attributes with different states will increase their respective scores by 1 (i.e. more likely to be predictive of class). If instead, both neighboring instances had the same class, referred to as a ‘hit’, then attributes with different states will get a decrease in score of 1 (i.e. less likely to be predictive of class). Relief-based algorithms are fast, and proficient at assigning higher weights to attributes with simple main effects, lower order epistatic interactions, and/or simple patterns of heterogeneity [19,20,18,43,41]. By default, ExSTraCS applies MultiSURF to discover EK, as it is the newest and most powerful of the algorithms. We refer readers to respective citations above for a complete description of each algorithm.

New to ExSTraCS, is the addition of Tuned-ReliefF (TuRF) [28,19]. TuRF is essentially a wrapper algorithm that can be applied to any of the four Relief-based algorithms to improve performance on datasets with larger numbers of attributes. The application of TuRF is particularly important in noisy problem domains, such as the one we consider in this paper. TuRF runs the chosen Relief-based algorithm repeatedly, each time systematically removing the worst attributes (*TuRF_Percent*), and re-estimating Relief algorithm weights in their absence. See [28] for a detailed description of the original algorithm. TuRF was originally conceived as a strategy to filter out the worst attributes and yield useful scores for the top remaining attributes. Within ExSTraCS we want to allow all attributes in the dataset an opportunity to be considered in rules. Therefore, in order to provide all attributes with scores and accurately reflect which attributes were removed earlier or later, we have made one alteration to the TuRF algorithm. As TuRF progressively removes *TuRF_Percent* of all attributes with each re-estimation cycle, it keeps track of which attributes each time. When the maximum number of re-estimation cycles ($\frac{1}{TuRF\_Percent}$) is reached, we calculate the difference between the highest and lowest scores (*diff*) of the remaining attributes. We then step backwards from the most recent set of removed attributes to the first. Beginning with the lowest score from best set of remaining attributes, each step we assign all attributes in the respective group a further discounted score than the mimimum thus far. This discount is 1% of *diff*. This way we can assign scores to all attributes and preserve useful information about the hierarchy of removed attribute scores. The ADM has been updated to manage the progressive removal of attributes for the TuRF wrapper.

Once raw EK scores have been loaded or discovered, a simple transformation is performed to ensure that all scores are within the range (0–1). This replaces the more complicated logistic transformation proposed in [43].

2.1.6 Rule Population—See (1) and (2) of Figure 1. This section begins a discussion of the incremental core algorithm that repeats over the course of learning. ExSTraCS will repeatedly cycle through all instances in the training data until I_{max} is reached. While the order of training instances were randomized when the dataset was loaded, ExSTraCS will cycle through these instances repeatedly in the same order. Preliminary analyses suggested that there was no benefit to repeated randomizations of the training instance order (data not shown). Within an iteration, a single training instance is passed to the rule population [P] which is the set of all unique rules. [P] is governed by a maximum population size parameter (*N*). When too many rules exist in the population the deletion mechanism is activated at the end of the iteration.

Here, we discuss rules and how they are represented within ExSTraCS. Rules in LCS algorithms follow an IF/THEN format, where the ‘IF’ represents a condition (i.e. to what attribute states does the rule apply?), and the ‘THEN’ represents what is traditionally referred to as the action (i.e. the class, endpoint, or phenotype). A given rule is only applicable to data instances that it’s condition matches, and can only adopt a single class endpoint as it’s action. For these reasons, an individual LCS rule can never be useful as a predictive model on it’s own.

In earlier work with LCS algorithms, (UCS [38], AF-UCS [37], and UCS-EK [43]) we applied a quaternary knowledge representation very similar to what had been used in other traditional LCS algorithms. This representation was applied to handle the genetic SNP datasets of our problem domain, where discrete genetic SNP attributes each had three possible states (0, 1, or 2). Figure 3 gives an example of this representation. Assuming that each value in the list is an attribute (A), and attribute order is zero-based numbering, the quaternary rule can be interpreted as follows [IF: (A1 = 2) and (A4 = 0) and (A6 = 1) and (A7 = 2), THEN: Class = 1]. Notice how the rule ignores the instance state values of any attribute with a ‘#’. This representation relied on the use of wild-cards (given by ‘#’) to indicate when the state of a given attribute could have any value (i.e. the attribute was generalized). Instead, ExS-TraCS adopts a mixed discrete-continuous attribute-list (DCAL) knowledge representation (see Figure 3) that permits attributes to be discrete and/or continuous-valued. The mixed representation example can be interpreted as follows [IF: (A1 = 2) and (A4 = 0) and (0.4 ≤ A6 ≤ 0.7) and (A7 = 'high'), THEN: Class = 1]. Notice that this representation accommodates instance state values that are integer, floats, or text. This strategy is largely similar to the one proposed by Bacardit in [3] which combined the attribute-list knowledge representation (ALKR), designed for continuous attributes, with the GABIL discrete attribute representation [16]. ALKR only stores attributes that are specified. This significantly reduces run time in both matching and attribute tracking [41]. This feature is particularly important in datasets with a large number of non-predictive attributes. ExSTraCS preserves the ALKR representation but avoids the GABIL representation in favor of a simpler but less generalizable strategy for representing discrete attribute states. In particular, a discrete attribute specified in the 'attribute reference' list (which stores the reference location for that attribute) can only have a single specified state value (0, 1, or 2) within the 'rule condition' (which stores the actual state value specified for that attribute). Differently, the GABIL representation allows multiple state values to be specified for a given attribute (e.g. the instance attribute state can have a value of either 0 or 2). While this may be advantageous for evolving a maximally compact rule-set, this approach is not in-line with the global approach to knowledge discovery proposed in [42] which relies on predictive attributes being specified more often across rules in the greater population. Additionally, the ExSTraCS representation yields individual rules that are arguably easier to interpret (i.e. there is less ambiguity within the IF/THEN statement) and they are likely to be more accurate individual predictors since they uniquely capture a more specific set of attribute states. The DCAL knowledge representation adopted by ExSTraCS has been applied within the covering, mutation, and crossover operators as described in [3].

A number of rule parameters are maintained and updated for each rule in the population over the course of learning. Table 2 summarizes these parameters including a brief description of what they are, how they're used, and when, if ever, there values are updated. The most important of these are *accuracy*, *fitness*, and *numerosity*. Accuracy, in the context of an LCS rule, is quite different than training or testing accuracy of the algorithm as a whole. Rule accuracy is calculated to be the proportion of instances in which a rule predicted the correct class out of those instances that the rule was able to match. To understand this further, consider that a rule could have a condition that only matched a single training instance, but made a correct prediction, meaning that the rule has an accuracy of 100%. While this rule is maximally accurate, we know that it is probably useless for making generalized predictions on testing instances that the algorithm has not yet seen. This hypothetical over-fit rule highlights why generalization pressures are critical to LCS algorithms.

Rule fitness is traditionally calculated as a power function of rule accuracy (i.e. *fitness* = *accuracy*) where the run parameter ν is typically set to 10. Previously, we observed that placing too much emphasis on optimal accuracy in calculating fitness led to dramatic overfitting in noisy problem domains [38]. Therefore ExSTraCS has adopted a default value of $\nu = 1$ making fitness equal to rule accuracy. The fitness of a rule proportionally impacts it's probability of being selected as a parent by the genetic algorithm, and is inversely proportional to it's probability of being deleted.

The numerosity of a rule is simply the number of copies of a given rule that have been introduced and preserved in [P]. Instead of individual copies of the same rule appearing in [P], Michigan LCS algorithms check to see of a rule already exists. If it does, it's numerosity is increased rather than adding that new rule copy to the population. Similarly if the algorithm seeks to delete a rule, the numerosity is decreased by 1. Once the numerosity of a rule reaches 0, the rule is removed entirely from the population. Rule numerosity plays a role throughout the algorithm, impacting the calculation of prediction votes, and deletion probabilities. When attempting knowledge discovery by manually inspecting the rule population, rules with the highest numerosity often have the best balance of accuracy and generality for making useful class predictions [48].

2.1.7 Match Set—See (3) of Figure 1. Getting back to the core learning cycle, a training instance is passed to [P] where the subset of rules that have a condition that matches the attribute states of the training instance are activated to form the match set [M]. All rules in [M] are relevant to the attribute states of the training instance, but may predict different class endpoints. ExSTraCS determines a rule to match by stepping through attributes specified in the attribute reference list. To match, all discrete instance states must equal the respective rule condition states. All continuous instance states must fall within the respective continuous intervals given by the rule condition. If an instance state has a *missing value* it will match anything, allowing ExSTraCS to handle the missing data point in a neutral manner. Because of this, ExSTraCS does not require imputation in the presence of missing data points. Like EK, imputation can add a positive or negative bias depending on it's quality. Imputation can still be performed prior to running ExSTraCS, but this is not currently built-in to the algorithm.

2.1.8 Correct Set—See (4) of Figure 1. ExSTraCS is designed specifically to deal with supervised learning problems, where the true class endpoint is known for each training instance ahead of time. As originally proposed in UCS, the match set [M] is divided into a correct set [C], and incidentally an incorrect set [I]. Being that we know the true class of the current training instance, [C] is simply the subset of rules in [M] that predict the 'correct' class. Intuitively, [C] represents the best rules currently in [P] for making a prediction on the current training instance. Later in the iteration, rules in [C] will receive a fitness boost for matching and making a correct prediction, while rules that matched, but made an incorrect prediction will be penalized with a fitness decrease.

2.1.9 Covering—See (5) of Figure 1. At this point it is possible that no rules in [P] have made it into [C]. For instance, this occurs the very first learning iteration, because [P] starts out empty. When [C] is empty the rule discovery operator known as *covering* is activated. Traditional Michigan LCS covering randomly generates a rule that both matches the training instance, and in the case of supervised learning assigns that rule the same class as the current training instance. Covering assures that at least one rule is included in the population that matches the current training instance, and serves to intelligently initialize [P] with rules that are sure to match at least one training instance in the search space. The 'random' generation of a rule condition is achieved by taking the attribute states of the training instance, and for each one deciding whether to keep that attribute state specified in the rule condition, or generalize it. For example, using the quarternary rule representation, covering might take the training instance [1,0,0,2,1,1,1]→1 and convert it into the following rule [#,0,#,#,1,#,#]→1, where only two attributes have been specified, and the rest have been generalized. The probability p_{spec} is applied to each attribute to decide whether it will be specified. In [43], covering was adapted to apply EK probabilities to bias the specification of attributes with high EK weights, and generalize attributes with low EK weights. This was successful in speeding up learning by initializing rules to specify attributes most likely to be useful in making accurate class predictions. The calculation of EK probabilities in that implementation relied on p_{spec} to determine the average number of attributes that would ultimately be specified in rules when applying EK.

As mentioned in the introduction, when the dataset includes large numbers of attributes, it is probable for rules to be generated that specify far too many attributes, which will lead to pure over-fitting, and prevent the algorithm from learning any useful generalizations. p_{spec} is a critical and difficult parameter to set especially as the number of attributes in the dataset becomes large. Ideally, we want the LCS algorithm to be able to discover rules that will match more than one training instance in the dataset. One potential solution is to set up a parameter sweep, where the LCS algorithm is initially tested with different values of p_{spec}. However this can be computationally expensive, time consuming, and makes running LCS algorithms more of a challenge. We could also purposefully set p_{spec} to an extremely low value, but this can lead to the frequent generation of completely general rule conditions that match everything, but also capture no useful information. Notably, ExSTraCS automatically prevents completely general rules from entering [P].

As a rule discovery mechanism, covering has been updated in ExSTraCS 2.0 to apply the RSL. This changes how covering operates both with and without the application of EK. Without EK, the new covering mechanism begins by randomly picking the number of attributes to specify (A_{Spec}) within the range (1, RSL). Next, ExSTraCS randomly selects A_{Spec} attributes to specify, where the new rule condition adopts the states of those attributes found in the training instance.

By default, ExSTraCS does apply EK to covering. In this situation, A_{Spec} is determined as before, but now ExSTraCS deterministically selects the A_{Spec} attributes with the top EK scores to specify. In a preliminary investigation we also tested an EK covering scheme which applied roulete wheel selection to probabilistically select the A_{Spec} attributes to specify. However, we found that this was both slower and less effective. With or without EK, both strategies assign the class of the current training instance to the rule.

Additionally, if an attribute is selected to be specified that has a missing value in the training instance, the attribute remains generalized in the rule. Also, if a continuous attribute is to be specified during covering, a continuous range centered around the current training instance attribute value is added to the rule condition [3]. The size of the interval is determined in the following way: (1) Get the difference *diff* between the maximum and minimum values observed in the training data for this attribute (previously computed and stored in the ADM). (2) Randomly select a range radius (r) that is between 25% and 75% of *diff*. (3) The interval is set to the value of the continuous attribute in the instance ±(0.5 × r). Pseudo-code for the covering strategy is given in Algorithm 1.

2.1.10 Rule Parameter Updates—See (6) of Figure 1. Now that [M], [C], and [I] have been formed, and at least one rule is in [C], rule parameters are updated for the rules that made it into at least one of these sets. Any rules in [P] that did not make it at least into [M] will not receive any updates during the current training iteration. Referring back to Table 2 and the column labeled 'Updated', the following parameters will get updated: (1) MatchCount; incremented for all rules in [M], (2) CorrectCount; incremented for all rules in [C], (3) Accuracy; recalculated for all rules in [M], (4) Fitness; recalculated for all rules in [M], (5) AveMatch-SetSize; an average estimate of the number of rules included in [M] (used to calculate deletion probability), and (6) TimeStampGA; gets set to current number of iterations that have passed when the GA is successfully activated (applied only to rules in [C]). Rule parameter updates are directly responsible for learning.

2.1.11 Subsumption—See (7) of Figure 1. Subsumption is a generalization mechanism that has been a part of most Michigan LCS algorithms since XCS [48]. ExSTraCS preserves subsumption as it was originally conceived, running the mechanism within [C] as well as between parent and offspring rules in the GA. Subsumption examines pairs of rules that have an accuracy greater than the parameter acc_{sub} (typically set to 0.99) and that have been involved in a [M] more than $theta_{sub}$ (typically set to 20). The more general rule (R_g) is a potential subsumer (e.g. [#,0,#,#,1,#,#]→1 using the quarternary rule representation for simplicity). The more specific rule R_s has the potential to be subsumed (e.g. [1,0,2,#,1,#,#] →1). If the above constraints were met and all specified attribute states in R_g are also specified in R_s then R_g subsumes R_s. This means that the numerosity of R_g increases by the numerosity of R_s, and then R_s is removed from the population.

While subsumption is currently preserved in ExS-TraCS it is unlikely that it is particularly effective in noisy problem domains where optimal rules are unlikely to achieve an accuracy of 0.99 or greater. This may prove an interesting target for future work.

2.1.12 Attribute Tracking and Feedback—See (8) of Figure 1. Attribute tracking and feedback mechanisms were first introduced in [37] and later implemented by ExSTraCS [41]. Attribute tracking (AT) is essentially a form of long-term memory for supervised, incremental learning (see (8) in Figure 1). Given a finite training dataset, a vector of AT scores is maintained for each instance in the data. During learning, the AT scores for the current training instance are increased based on which attributes are specified in rules found in [C]. Post-training, these scores can be applied to characterize patterns of association in the dataset. In particular, heterogeneous patterns can be sought that may capture clinical patient subgroups that in turn may be targeted for research, treatment, or preventative measures [37]. Note that using AT alone does not impact ExSTraCS learning performance. Attribute feedback (AF) is applied to the GA mutation and crossover operators. AF probabilistically directing rule generalization based on the AT scores from a randomly selected instance in the dataset. The probability that AF will be used in the GA is proportional to the algorithm's progress through the specified number of learning iterations (i.e. AF is applied infrequently early-on, but frequently towards the end). AF speeds up effective learning by gradually guiding the algorithm to more intelligently explore reliable attribute patterns. These mechanisms and their application are further detailed in [37, 40,41]. One final observation that should be made in the context of scalability is that AT and AF are likely to become somewhat difficult to apply as datasets become extremely large, specifically when there are huge numbers of training instances or potentially predictive attributes. However in the context of this investigation, we do not consider large enough data for this to be a problem.

2.1.13 Genetic Algorithm—See (9) of Figure 1. The next step in the algorithm is to apply the second rule discovery mechanism. Specifically, ExSTraCS applies a niche GA that selects two parent rules from [C] and then generates two offspring rules to be added to [P]. The GA is activated if the difference between the current iteration and the average *TimeStampGA* of rules in [C] is greater than the run parameter $theta_{GA}$ (typically set to 25 by default). This prevents the GA from running very early on, or on sets of rules in which many of the rules were recently involved with the GA mechanism. This helps the algorithm to distribute its efforts exploring the search space.

When activated, the GA begins by selecting two parent rules. ExSTraCS has both the roulette wheel and tournament selection implemented as options for parent selection. Tournament selection is used by default, as this strategy was found to be more parameter independent, noise independent and efficient when implemented in the XCS algorithm [15].

Next, a uniform crossover operator is applied with probability $\chi = 0.8$ which seeks to shuffle the conditions of the two parent rules, recombining them into two new offspring rules. AF is implemented into the ExSTraCS GA as previously described in [37]. In brief, the AT scores from a randomly selected instance are used to bias the swapping of attributes between parent rules, encouraging attributes with high AT values to end up in one rule, and attributes with low AT values to end up in the other. The intuition here is that high scoring attribute combinations identified for one instance, will be more likely to be tested out in another. Crossover proceeds as previously implemented in [41]. However crossover can yield an offspring rule that specifies more attributes than the RSL allows. In order to maintain this limit, the algorithm checks to see if the number of attributes specified in either offspring rules is greater than the RSL. If AF is active, attributes with the lowest AF scores are generalized from the rule until the RSL is reached. Alternatively if AF is not active, attributes are randomly selected to be generalized from the rule until the RSL is reached.

Lastly, a mutation operator is applied to both off-spring rules. Previous implementations of mutation in Michigan LCS algorithms apply a default probability of $\upsilon = 0.04$ for every attribute in the dataset, potentially 'mutating' this attribute in the rule condition. LCS mutation is somewhat different than other evolutionary algorithms. Specifically if an attribute was originally specified in the rule, mutation generalizes it to 'wild/don't care'. If the attribute was originally not specified in the rule, it now specifies the attribute to have the current attribute state of the training instance. As a result, new offspring rules generated by the GA will always match the current training instance under consideration. This type of mutation becomes problematic when scaling up to data with large numbers of attributes as discussed in section 1, since random chance will tend to mutate rules towards having roughly half of all attributes specified. ExSTraCS 2.0 updates the mutation operator in two ways: (1) adapting it to the RSL and (2) integrating EK to guide mutation in the context of the RSL.

First, we determine the final number of attributes A_{FSpec} that will be specified in the rule following mutation. The parameter υ (which notably does not have the same function as in the traditional mutation operator) is used to pick some number of *steps*. Specifically we repeatedly choose a random number in the range (0,1) and increment *steps* each time until this random number is less than υ. Assuming a low value for υ (a default value of 0.04 is preserved), *steps* will typically equal 0. Next we randomly pick A_{FSpec} from the range (A_{CSpec} − *steps*, A_{CSpec} + *steps*) where A_{CSpec} is the current number of attributes specified in the rule. Also, keep in mind that this range cannot go below 1 or above RSL. This forces ExSTraCS to search the problem space in which we are most likely powered to find useful generalizations.

At this point we compare A_{Spec} to the current number of attributes specified in the rule. Now, one of the following scenarios is possible: (1) $A_{FSpec} = A_{CSpec}$, i.e. specificity is maintained, (2) $A_{FSpec} > A_{CSpec}$, i.e. specificity is increased, (3) $A_{FSpec} < A_{CSpec}$, i.e. specificity is decreased. If specificity is to be maintained ExS-TraCS retains a probability υ that the mutation operator will abort, and the rule will remain unchanged. Alternatively, ExSTraCS mutates two attributes in the rule (one specified, and another generalized) so that overall specificity is unchanged. If specificity is to be increased, ExSTraCS specifies attributes until A_{FSpec} is reached. If specificity is to be decreased, ExSTraCS generalizes attributes until A_{FSpec} is reached.

This new mutation scheme applies the opportunity to utilize AF, EK, or simply rely on random chance to select attributes to be mutated. First, a target attribute is selected as follows: If EK is active (as it is by default), ExSTraCS applies a 50% chance that EK weights will be applied to roulette wheel selection to pick the target attribute. If selecting an attribute to specify, a larger attribute EK weight means a larger probability of being selected. If selecting an attribute to generalize, a smaller attribute EK weight means a larger probability. If EK is deactivated, or the 50% chance opts to avoid the use of EK, the target attribute is chosen randomly. This 50% chance of using EK to choose the mutated attribute(s), assures opportunity for rule diversity by limiting the application of EK bias.

Once an attribute has been selected as the target for mutation, we check to see if AF is currently active. If so, the AT scores from a randomly selected instance are used to probabalistically allow the attribute to be specified if the AT score is high, or prevent it if low. If generalizing an attribute, the opposite is true. Lastly, there is a probability of υ that the class endpoint of a given rule will be mutated to a different class.

Before the two offspring classifiers are added to [P], ExSTraCS checks to see if either rule already exists in the population (if so that rule's numerosity is increased). Also before adding offspring rules to [P], ExS-TraCS attempts to perform GA subsumption between parent and offspring rules as discussed in 2.1.11.

2.1.14 Deletion—See (10) of Figure 1. At the end of a learning cycle ExSTraCS sums the number of rules currently in [P] (*microPopSize*). The micro-population size is the sum of all rule numerosities in [P]. The macro-population size is the number of unique rules currently in [P]. If *microPopSize* is greater than *N*, than rules will be selected for deletion until *N* is reached. Rules are selected for deletion using roulette wheel selection. The calculation of rule deletion weights is inherited from XCS [48] and UCS [9]. Specifically the deletion weight of a given rule is calculated as follows:

$$deletionWeight = \frac{aveMatchSetSize * numerosity^2 * meanFitness}{fitness} \quad (2)$$

Here, *meanFitness* is the average Fitness of all rules in [P] while the other rule parameter values in the above equation are specific to the given rule. To prevent division errors, if fitness = 0, the run parameter *initFitness* (set to 0.01 by default) is used instead of fitness. Additionally, to protect rules from deletion that have above average fitness and low numerosity (i.e. If $\frac{fitness}{numerosity} \geq \delta * meanFitness$), or similarly protect rules with very little experience (i.e. If *matchCount* < $theta_{del}$), an alternate calculation of *deletionWeight* that will yield smaller weights is applied.

$$deletionWeight = aveMatchSetSize * numerosity \quad (3)$$

2.1.15 Rule Compaction—See (C) of Figure 1. Once ExSTraCS has reached I_{max}, we are left with a [P] that ideally has learned some accurate generalizations that will be useful for making class predictions. When Michigan LCS learning stops, some of the rules will inevitably be inexperienced as the algorithm is always testing new areas of the search space, while seeking to preserve the best, more experienced rules. Some of the rules in [P] are useless artifacts of the rules discovery process that contribute nothing, or even hinder, accurate predictions. Additionally, a certain amount of rules that overlap in the search space are likely to exist in [P]. To remove these poor or overlapping rules, and/or to reduce the overall size of [P] with the intent of increasing interpretability via manual rule inspection, ExSTraCS makes six rule compaction strategies available to post-process [P] and yield a compacted population [P_c] (see (C) in Figure 1). Each rule compaction strategy was implemented and evaluated in [35]. Comparisons in [35] suggested that the simple Quick Rule Filtering (QRF) was both the fastest, and most well suited to the theme of global knowledge discovery [42] in which it is more important to preserve or improve performance than to minimize rule population size. ExSTraCS applies QRF by default. The remaining subsections discuss other key features that highlight how the evolved population of rules [P_c] can be applied and evaluated.

2.1.16 Prediction—See ‘Prediction’ in Figure 1. As a supervised learning algorithm, iterations of ExSTraCS do not alternate between an explore/exploit phases as described in XCS [48] and other reinforcement learning-driven LCS algorithms. In the classic exploit phase, a prediction array is generated using [M] to obtain a class prediction. ExS-TraCS will generate a prediciton array in the following situations: (1) performance tracking during learning (ExSTraCS generates a prediction array every iteration to track estimated prediction performance), (2) evaluation of rule population prediction accuracy over a complete training or testing dataset, and (3) applying [P] as a prediction machine to other previously unseen instances.

ExSTraCS adopts a prediction strategy similar to XCS with a couple minor extensions to increase the chances that a class prediction can be made without resorting to a random choice between class (if class votes are tied). The 'vote' calculated for each endpoint class is the sum of all numerosity-weighted fitnesses (*numerosity*fitness*) for rules specifying a given class in [M]. The class with the largest vote is selected to be the predicted class. ExSTraCS extends this approach, also similarly tracking the sum of numerosities and the sum of *initTimeStamp* for each class. If there is a tie for best class after the initial vote, then this tie is broken by the class with the greatest numerosity sum (i.e. more rules in [M] predict the given class). If a tie persists then the class with the younger average set of rules is chosen as the prediction since all things being equal, we have the expectation that newer rules should be better. If the tie still cannot be broken, a random class is selected as the prediction. Ties at any level in prediction are rare, thus we don't expect this expansion to have much influence in most problem domains. It does however seem appropriate to use any information available to avoid making random predictions.

If [M] is empty when the prediction array is activated, then no rules cover the current data instance. When this occurs, no prediction can be made, as [P] is said to not 'cover' the given instance. ExSTraCS tracks and reports this coverage statistic for complete evaluations of training or testing data

2.1.17 Output—See 'Output' in Figure 1. Upon request, ExSTraCS will yield up to five distinct output files when I_{max} has been reached. Additionally, the user can specify any number of checkpoints for learning to be paused, and complete evaluations of [P] to be completed yielding the same set of output files. Output files include (1) the population of rules [P] including rule parameter values for respective rules, (2) population statistics, summarizing major performance statistics including global training and testing accuracy of the classifier population [42], (3) co-occurrence scores for the top specified pairs of attributes in the dataset [42] (discussed further in section 2.2.2), (4) attribute tracking scores for each instance in the dataset [37], and (5) a summary of class predictions for the testing datasets. This last output files is new to ExSTraCS 2.0. It includes the class prediction and associated class votes for each instance in the testing data. This allows the solution evolved by ExSTraCS to be explicitly tested and evaluated as a prediction machine. Class votes for each instance could be utilized as fuzzy predictions, Bayesian probabilities, or estimates of prediction confidence. These output files facilitate algorithm transparency, interpretation, and knowledge discovery. For a detailed exploration of how knowledge discovery or statistical significance testing might be pursued in ExSTraCS or similar Michigan LCS algorithms we refer readers to [42].

2.1.18 Miscellaneous—Here we review some of the additional features implemented in ExSTraCS to facilitate flexibility of use. First, in addition to a training dataset for learning, ExSTraCS optionally allows a testing dataset to be loaded so that the algorithm may be evaluated on instances it has not yet been exposed to. Notably, whenever evaluations of the algorithm are being completed, learning and rule discovery is turned off so that the rule population remains unchanged. ExSTraCS is designed to facilitate cross-validation analyses, where an original training dataset is broken into *n* parts, and the algorithm is independently trained *n* separate times on $\frac{n-1}{n}$ of the data, with a different $\frac{1}{n}$ of the data reserved for testing each run (*n* = 10 by default). Cross-validation analysis allows for an assessment of over-fitting, and the ability of the algorithm to make useful predictions on instances it has not yet seen. Optionally, ExSTraCS allows the user to generate cross-validation datasets from an original training dataset, and to serially run the algorithm *n* independent times, each yielding associated output files.

Next, ExSTraCS offers a 'reboot' option, which allows the user to load any previously generated rule population output file, and pick up training from where it had previously left off. This allows the user to add additional training iterations without having to run ExS-TraCS from scratch.

Also, class imbalance can bias the standard calculation of prediction accuracy, such that it does not meaningfully reflect performance. ExSTraCS applies a balanced accuracy calculation to all complete evaluations of [P]. Balanced accuracy (*balAcc*) is the average of sensitivity and specificity. ExSTraCS extends the calculation of balanced accuracy described in [47] to accommodate discrete endpoints with more than two classes. This is done by tracking whether a prediction is a true positive (TP), true negative (TN), false positive (FP), or false negative (FN) for each class. Using these values we calculate sensitivity and specificity for each class using the following equations:

$$sensitivity=\frac{\sum TP}{\sum TP+\sum FN} \quad (4)$$

$$specificity=\frac{\sum TN}{\sum TN+\sum FP} \quad (5)$$

Ultimately, balanced accuracy is calculated as follows, where *n* is the number of classes.

$$balAcc=\frac{\sum_{i=1}^{n}\frac{sensitivity_i+specificity_i}{2}}{n} \quad (6)$$

2.1.19 Logistics—Table 3 summarizes all of the ExSTraCS learning parameters and the default values applied in this study. The ExSTraCS algorithm is run from the command-line and coded in Python to promote readability and facilitate ongoing development. The code is open source, well annotated and freely available at *sourceforge.net* [36]. The software comes with a complete users guide that details the parameters found in Table 3, reviews how to apply the features built into ExSTraCS, and when users may want to turn off certain mechanisms (e.g. attribute tracking, attribute feedback, expert knowledge, or rule compaction).

2.2 Experimental Evaluation

In this section we detail the experimental evaluation applied to test the scalability of ExSTraCS. This includes a description of the simulation studies, experimental comparisons, and an overview of the different performance metrics applied. In previous work, we found that UCS outperformed XCS [38], and two other Pittsburge-style LCS algorithms [39] on complex simulated SNP datasets within which patterns of pure, strict epistasis and genetic heterogeneity were modeled. Later, we also demonstrated that the use of attribute tracking and feedback in AF-UCS [37] as well as the application of expert knowledge in UCS-EK [43] outperformed UCS on the same datsets. Most recently, we introduced ExSTraCS 1.0 which combined AT, AF, EK, rule compaction, and the rule representation previously described [41]. Evaluation of ExSTraCS 1.0 on a similar set of simulated datasets concluded that the combination of these features significantly improved or preserved all performance metrics when compared to implementations that applied these mechanisms independently. In the present study, ExSTraCS 1.0 serves as the standard of comparison with which we evaluate the impact of the RSL, TuRF, and expert knowledge mutation in ExSTraCS 2.0.

2.2.1 Simulation Studies—This study included two separate complex genetic simulation studies, and a study of different benchmark multiplexer problems. The first simulation study is identical to the one performed in [41], including a total of 960 diverse datasets with discrete SNP attributes. These datasets range in difficulty from those that we know ExSTraCS 1.0 can reliably solve to those with massive amounts of noise and particularly challenging patterns of association that are currently unsolvable. All simulated datasets, and the associated simulated genetic models used to produce them were generated using GAMETES [45], an open source piece of software designed to generate a diverse spectrum of pure, strict epistatic models. A pure, strict interaction is one in which examining a single attribute or subset of attributes involved in the interaction model will yield no information about the class endpoint (case or control). This is a particularly challenging pattern to detect. Two-locus epistatic genetic models with heritabilities of (0.1, 0.2, or 0.4) and attribute minor allele frequencies of 0.2 were simulated in GAMETES applying a model difficulty estimate to choose model architectures that were maximally and minimally difficult to solve (i.e. two models were selected for each heritability) [44]. For reference, a heritability of 1 would be an example of data without noise. In theory, the best testing accuracy we can hope to reliably achieve for a dataset with heritability = 0.4, is 0.7. For a dataset with heritability = 0.1 this drops to only 0.55, making these particularly noisy datsets.

In turn, these models were used to generate an archive of simulated datasets that also included patters of genetic heterogeneity (i.e. one part of the dataset was generated to model epistasis in one pair of attributes, and the other with a different pair of epistatic attributes). In this way, our simulated datasets concurrently models patterns of both epistasis and heterogeneity. All datasets in this first simulation study include a total of 20 attributes. Four of the attributes were predictive as two separate epistatic interactions heterogeneously related to the endpoint, and the remaining 16 were randomly generated, non-predictive attributes. Sample sizes (i.e. number of instances) of 200, 400, 800, or 1600 were generated each with a heterogeneous mix ratio of either (50:50 or 75:25) (e.g. 75% of instances were generated from one epistatic model, and 25% were generated from a different one). 20 replicates of each model and dataset combination were generated yielding a total of 960 datasets (i.e. 3 heritabilities × 2 model difficulties × 4 sample sizes × 2 mix ratios × 20 replicates). Furthermore we applied 10 fold cross validation in the assessment of every dataset, which means that each algorithm version was run a total of 9600 times for this simulation study. For each, ExSTraCS was run up to I_{max} = 200, 000.

Our second simulation study exclusively examines scalability. Due to the additional computational demands, this study was limited to a single combination of model and dataset features. Specifically we chose a model/dataset combination from the first simulation study that we knew ExSTraCS could reliably learn when only 20 attributes were included in the dataset. This combination involved a heritability of 0.4, a minor allele frequency of 0.2, the maximally easy model architecture, a sample size of 1600, and a mix ratio of 50:50. 20 replicate datasets with this combination of contraints were generated having either 200 or 2000 attributes. 10-fold cross validation was applied so that each algorithm version was run a total of 200 times (20 replicates × 10 CV). Furthermore in this second simulation study we also evaluated performance when ExSTraCS was run for a larger number of iterations, as well as with a larger population size.

Our third study explores a spectrum of multiplexer problems (i.e. 6-bit, 11-bit, 20-bit, 37-bit, 70-bit, and 135-bit). An *x*-multiplexer is a combinational logic circuit that takes *x* binary inputs, where $x = k + 2^k$, and gives one binary output. The value encoded by *k* address bits gives the position of one of the remaining 2^k register bits. The binary value at that target register bit is given as the output value. For example, in the 6-bit mulitplexer, if the input is $\underline{11}001\mathbf{0}$ then the output will be 0 since the underlined address bits point to the last bit (i.e. index 3 in base ten, highlighted in bold with value 0). The 6-bit multiplexer has itself has been referred to as a difficult problem, since it and all other standard multiplexer problems are epistatic, heterogeneous (i.e. multimodal), and have no reliable main effects (i.e. no linear relationships between single attributes and classification). Multiplexer problems also conveniently scale up to higher dimensions, which require the discovery of a set of optimally generalized rules to solve the problem and make reliable output predictions. Table 4 summarizes major characteristics of the 6-bit to 135-bit multiplexer problems, including the order of epistatic interaction, the number of heterogeneous feature combinations, the number possible unique training instances, and the minimum number of accurate, optimal rules required to solve the respective problem. The required number of optimal rules [O] given in this table assumes that we are searching for a best-action map rather than a complete-action map, the later being applied to reinforcement learning LCS algorithms such as XCS [11]. Multiplexer problems are most commonly used as benchmark problems applied throughout the LCS literature since their introduction by Wilson in [49]. The 6-bit and 11-bit multiplexers have been explored using other machine learning strategies such as genetic programming [21,22], decision trees [32,7], and neural networks [8,34] however they have rarely appeared outside of the LCS literature in recent years, presumably, in-part because other machine learning algorithms have trouble adapting to these nonlinear, distributed types of problems, particularly when they are scaled up to a higher difficulty. While a handful of studies have reported successfully tackling the 70-bit multiplexer problem [13,12,11,14,4,10,23], to the best of our knowledge, only two studies to date have claimed to tackle the 135-bit multiplexer problem [11, 23], both of which involve LCS algorithms. However, in [11], Butz successfully solves the 'layered' 135-bit multiplexer, as opposed to the standard 135-bit problem. A 'layered' multiplexer provides additional fitness guidance towards specification of address bits in rules from the beginning. This greatly simplifies the problem by essentially giving part of the solution to the algorithm from the start (i.e. the address bits should be specified in optimal rules). In the second study [23], Iqbal *et. al.* successfully solved the standard 135-bit multiplexer, but used building blocks learned from training on simpler multiplexer problems in order to achieve scalable learning. This approach requires the unlikely assumption that when faced with a complex problem, a similarly structured simpler versions of that problem will be available from which to derive reliable building blocks. In real world classification problems such as those found in bioinformatics this would be an unrealistic assumption. Therefore, to the best of our knowledge, the standard 135-bit multiplexer problem has not yet been reported to have been solved as a stand-alone problem by any LCS or other machine learning algorithm. In the present study we have decided to investigate multiplexer problems as they are characteristic of our target complicating phenomenon (i.e. epistasis and heterogeneity), and to put the performance of ExS-TraCS into a problem context more familiar to LCS and machine learning researchers.

ExSTraCS analysis of the 6 different multiplexer problems was run with a few differences compared to our previous simulation studies. For the most part, run parameters given in Table 3 were kept the same with the exception of N, I_{max}, and ν. Given that these multiplexer problems did not include noise, we used a more typical LCS algorithm setting of $\nu = 10$ such that ExSTraCS would give a greater fitness weight to rules with a very high accuracy. Table 5 gives the settings for N and I_{max} used to solve each of the multiplexer problems. Maximum rule population sizes were selected based on those used in [23] which has reported solving the different multiplexer problems in fewer learning iterations than other Michigan-style LCSs, and thus will be our standard for comparing multiplexer results. Different than any other attempt to solve the multiplexer problem, Table 5 also gives the finite number of randomly generated multiplexer training instances used to train ExSTraCS on a respective multiplexer problem. Previous analyses of multiplexer problems learn online from an essentially infinite stream of randomly generated muliplexer instances that serve to train or test the algorithm's predictive performance as part of an explore/exploit scheme, respectively. As a supervised learning LCS, ExSTraCS only performs explore iterations, constantly learning on training instances and avoiding wasteful estimation of predictive performance. Instead, ExSTraCS periodically pauses learning, and separately evaluates the training vs. predictive performance of the current [P] on the entire training and testing dataset (i.e. two batch-wise evaluations). Another reason for this type of finite multiplexer learning, is that ExSTraCS requires a finite training dataset in order to generate expert knowledge weights to drive covering and mutation mechanisms. Having a finite training dataset is also more in-line with typical real-world supervised learning applications. Also we would expect training on a finite set of instances to be harder than a constant stream of new training examples, since the latter will provide more unique examples from which to enforce rule generalizations. Training dataset sizes were chosen somewhat arbitrarily and through trial and error for the harder problems. 30 randomly generated multiplexer datasets for each of the 6 different multiplexer problems were trained using ExSTraCS. In each case, one of the other 30 multiplexer datasets was used for testing ExSTraCS predictions. It should be pointed out that every simulated multiplexer datset is comprised entirely of predictive attributes, where in our previous simulation studies only 4 attributes were predictive, and the rest were non-predictive. For example, in the 135 multiplexer datasets, all 135 attributes are predictive of class endpoint and crucial to solving the complete problem.

All simulated datasets can be made available upon request or similar simulated genetic datasets may be generated using the GAMETES software package [45, 44]. Pair-wise statistical comparisons between the ExS-TraCS implementations were made using the Wilcoxon signed-rank tests. We consider statistical significance at a p-value below the 0.05 cutoff. We also correct for multiple statistical comparisons using a very conservative Bonferroni correction based on 20 separate algorithm comparisons involving 9 performance metrics. A maximally conservative correction including all families of hypothesis would thus correct for 180 separate tests yielding a significance cutoff of 0.00027. A slightly less conservative approach would treat the 9 performance metrics statistic as a separate families of hypothesis yielding a significance cutoff of 0.0025. All three cutoffs are provided. All statistical evaluations were completed using R. All analyses were performed using ‘Discovery’, a 2400 core Linux cluster available to the Dartmouth College research community.

2.2.2 Performance Metrics—In order to thoroughly evaluate performance, we consider a variety of standard and customized metrics that we have applied in previous work [42,37,43,35,41]. These include training and testing accuracies which have been determined by running the static rule population as a prediction machine on all instances in the respective datasets. Balanced accuracy, described in section 2.1.18 is used to calculate these respective accuracies. Testing accuracy is the metric that we particularly want to pay attention to when evaluating if ExSTraCS is learning useful generalizations. We also examine the average rule generality of [P]. Average generality is weighted by the numerosity of rules. This metric gives the average proportion of attributes that are unspecified in rules across the population. Generality is useful for evaluating the algorithm’s progress in maximizing rule generality while balancing accuracy. All other metrics being equal, a higher generality is preferable. Next, we consider the macro-population size (i.e. the number of unique rules in [P]). A smaller macro-population size is generally considered to be more interpretable in the context of manually inspecting the rules. This metric might also be a useful indicator that the population size may need to be increased in order to improve performance. Specifically, if macro-population size is equal or close to the micro-population size, this suggests that the best, most experienced rules are not getting the opportunity to have their numerosities increased, which helps to protect them from complete removal from the population. Also, we include run-time (i.e. cpu-time) as a metric. Reported run-times exclude pre-processing of EK scores. Efficient implementations of the Relief-based algorithms such as the ones included with MDR [29] can make this pre-processing a relatively trivial contribution to run-time. However it should be noted that the current Python implementations of Relief-based algorithm built into ExSTraCS are needlessly slow, albeit effective.

Along with these standard LCS metrics, we have previously introduced a set of four conservative 'power' metrics designed to quantify the interpretability of [P] using global trends as described in [42]. Each of these metrics are based on the fact that we are working with a simulation study where we know the true underlying model, and the algorithm is tasked with finding it for itself. Specifically, 'Both Power' is the proportion of datasets in which the four predictive attributes were specified more frequently in the rules of [P] than any others. This gives us a sense of the algorithm's ability to correctly identify both two-locus, heterogeneous models. 'Single Power' is the proportion of datasets in which the attributes of at least one of the two separate, two-way epistatic models were specified more frequently in the rules of [P] than any others. This gives us a sense of the algorithm's ability to find at least one of the two epistatic models. The last two metrics are based on attribute co-occurrence in rules. This involves the examination of pairs of attributes that are specified together in rules across the population. This is intended to give us insight into the nature of attribute interactions, and facilitate the distinction between an epistatic interaction and a heterogeneous relationship when performing knowledge discovery. 'Both Co-Power' is the proportion of datasets in which the two pairs of modeled epistatic interactions are also the two pairs of attributes with the highest co-occurrence counts in rules of [P]. 'Single Co-Power' is the proportion of datasets in which at least one of the epistatic pairs has the highest co-occurrence count in rules of [P]. These last metrics help to identify if the underlying pattern of heterogeneity can be identified during knowledge discovery. The most important performance metrics considered in this study include testing accuracy, and the various measures of power.

3 Results and Discussion

We begin by reviewing the results of the first simulation study that examines a full spectrum of complex noisy datasets with 20 attributes (see Tables 6 through 8). Each table summarizes performance metrics averaged over 960 datasets (each run 10 times for cross validation) with statistical comparisons between versions of ExSTraCS. In order to properly evaluate the RSL, we want to give ExSTraCS 1.0 the best opportunity to perform well by adjusting the run parameter p_{spec} to specify a similar number of attributes in covering to what RSL would allow. In previous work we have typically used $p_{spec} = 0.5$ on datasets with 20 attributes [38,39,37,43,40,35,41]. This simulation study includes datasets with 200, 400, 800, and 1600 instances. Referring back to Table 1 we can identify the RSL that will be used by ExSTraCS 2.0 for each dataset size. Given that all simulated SNP datasets have $\varepsilon = 3$, ExSTraCS 2.0 will use RSL's of 5,6,7, and 7 for respective numbers of training instances. Given a RSL of 7, covered rules in this first study will, on average, specify 3.5 attributes. 3.5 attribute out of a total of 20 is 0.175. Table 6 gives the first comparison between two separate analyses of ExSTraCS 1.0 where the parameter setting of $p_{spec} = 0.5$ is compared to $p_{spec} = 0.175$. Here we observed few significant differences between a p_{spec} of 0.5 vs. 0.175. However, using 0.175, ExSTraCS 1.0 required a significantly shorter run time, and yielded a small significant increase in average rule generality. Therefore, ExSTraCS 1.0 with $p_{spec} =$

0.175 becomes our standard for comparison. Notably we will use the same logic for setting p_{spec} in ExSTraCS 1.0 when we scale up to larger datasets (i.e. a p_{spec} of 0.0175 is used for datasets with 200 attributes, and a p_{spec} of 0.00175 is used for datasets with 2000 attributes).

Based on the findings in [43], we began with the understanding that using EK within the mutation operator might add too much bias towards the highest EK scoring attributes. This had previously been demonstrated to lower 'Both Co-Power' in similar datasets with 20 attributes. This translates into a loss in our ability to distinguish patterns of heterogeneity. Therefore, we next compared ExSTraCS 1.0 (p_{spec} = 0.175) to a preliminary implementation of ExSTraCS 2.0 that includes RSL, but excludes the use of EK within the mutation operator (see Table 7). ExSTraCS 2.0 with RSL yields significant and dramatic performance improvements for 'Both', and 'Single' Power as well as a small significant increase in average testing accuracy and 'Single Co-' Power. However, notably we observed an increase in run time as well as a loss in 'Both Co-Power'. Clearly, in this smaller-scale simulation study, the RSL leads to more efficient learning, as it limits the dimmensionality of the rule search space preventing the algorithm from spending time considering rules that are far too over-specific. Regarding the loss in 'Both Co-Power'; manual rule inspection of a number of rule populations evolved using RSL indicated that rules specifying all 3–4 of the modeled predictive attributes were much more common. Since this is simulated data, we know that optimal rules (in the context of knowledge discovery) would specify epistatically interacting pairs alone. However, given the noise modeled in these datasets, the accuracy of rules that specify these pairs alone will never be 'optimal' (i.e. accuracy = 1). Ultimately, the loss in 'Both Co-Power' does not indicate a problem with the RSL, but rather it points to a fundamental issue involving the calculation of rule fitness in noisy problem domains. Specifically, when the fitness of a single rule is based on accuracy alone, it has no concept of what it means to be a 'fit' rule in the context of the population as a whole. Going back to our manual rule inspection, rules dominate which are slightly over-specified, and have an accuracy of 1 or close to 1. As a result, it is not surprising that we see a loss in our very conservative 'Both Co-Power' which requires that the two independent pairs of interacting attributes are specified together in rules more than any other combination of attributes. All it takes for loss in this metric is for one other attribute pair to be co-specified in rules more frequently. In this simulation study, it turned out that some other combination of the four predictive attributes (X0, X1, X2, and X3) was typically one of the top two co-specified. For instance we may want attributes X0 and X1 and separately X2 and X3 to be co-specified the most (as they constitute the two heterogeneous epistatic pairs), but instead we might see X0 and X1 as well as X0 and X2. We plan to address the issue of fitness calculation in the context of noisy problems as part of the ongoing development of ExS-TraCS.

Next, we added our new EK mutation strategy to ExSTraCS 2.0 and compared it to our previous ExS-TraCS 2.0 implementation (see Table 8). Later we will see why the addition of this mutation mechanism to ExSTraCS 2.0 was important to scalability. After the incorporation of EK mutation we observed further dramatic and significant increases in 'Both' and 'Single' Power, as well as average testing accuracy. We also observed a significant decrease in run time, however compared to ExSTraCS 1.0, this run time is still significantly higher. The significant and dramatic loss in 'Both Co-occur Power' can be attributed to the same phenomenon linked to fitness that was previously discussed, but this effect is exacerbated by the use of EK in mutation which will directly impact the overall frequency of attribute specification in [P]. It is also worth pointing out that ExSTraCS 2.0 with EK Mutation has roughly reached the estimated optimal average testing accuracy for this set of 960 datasets (≈ 0.6167). This value is estimated from the heritabilities of the simulated datasets. An equal number of datasets have a heritability of either 0.4, 0.2 or 0.1, which should yield maximum testing accuracies of approximately 0.7, 0.6, or 0.55 respectively (averaging to ≈ 0.6167).

To further characterize performance differences between ExSTraCS 1.0 and ExSTraCS 2.0 with EK mutation, we explore performance differences when the 960 datasets of this first simulation study are segregated by their heritability, sample size, architecture difficulty (i.e. hard (H), or easy (E)), and mix ratio. Figures 4, 5, and 6 give boxplot summaries of testing accuracy, 'Both Power', and 'Single Power', respectively comparing ExSTraCS versions 1.0 and 2.0. Keep in mind that 'Sample Size' corresponds to the number of unique training instances in the original (pre-cross validation) datasets, and that heritability is inversely proportional to noise in the data. Note that the values summarized by each individual boxplot correspond to 20 replicate datasets evaluated using cross validation. We can see from these figures, that ExSTraCS 2.0 with EK mutation outperforms version 1.0 for nearly every dataset combination.

Now that we have demonstrated the efficacy of ExS-TraCS 2.0 on a diverse array of smaller, complex noisy datasets, we move on to the second simulation study specifically examining scalability in a subset of similar datasets with either 10 or 100 times as many attributes. Tables 9 through 11 examine performance on 20 select datasets with 200 attributes and a heritability of 0.4. In these datasets, the maximum average testing accuracy we can hope to achieve is 0.7.

Preliminary studies indicated that the quality of EK scores became particularly important to successful learning in datasets with a greater number of potentially predictive attributes (not shown). TuRF was designed to improve the quality of EK scores, particularly in larger noisy datasets [28]. Table 9 explores the impact of using TuRF in combination with MultiSURF vs. MultiSURF alone to generate and apply EK scores to learn on datasets with 200 attributes using the original ExSTraCS 1.0 algorithm. It is worth pointing out that the introduction of TuRF alone to the analysis significantly improves testing accuracy as well as one of the power metrics. However this alone is not sufficient for ExSTraCS to reliably find the underlying pattern of association in the presence of 196 non-predictive attributes.

Next we compared ExSTraCS 1.0 ($p_{spec} = 0.0175$) to ExSTraCS 2.0 without EK mutation and without TuRF in order to examine the effect of RSL alone. Notice that the ExSTraCS 1.0 is able to achieve a testing accuracy well above 0.5, indicating that has been able to learn useful generalizations. However, in terms of our conservative power metrics, ExSTraCS 1.0 fails to scale up performance in clearly identifying the 4 true underlying predictive attributes out of the 200 in the dataset. However with the introduction of RSL in ExSTraCS 2.0, we observe very dramatic and significant performance improvements for all key metrics as well as a significant decrease in run time. Clearly, RSL alone is effective at scaling ExSTraCS performance from 20 to 200 potentially predictive attributes.

As before we added the new EK mutation mechanism back into ExSTraCS 2.0 and compared it to ExS-TraCS 2.0 without EK mutation (see Table 11). We observed a small non-significant increase in testing accuracy, a significant increase in rule generality, and a significantly more compact [P]. There was no room for improvement in any of the power metrics. The importance of the EK mutation mechanism only becomes clear within even larger datasets. We also made the same comparisons presented in Tables 10 and 11 adding back TuRF (not shown) and observed very similar results, with only slight, non-significant improvements to testing accuracy and a significant reduction in macro population size.

The last part of this second simulation study examines 20 datasets with 2000 attributes. Again we compare ExSTraCS 1.0 (p_{spec} = 0.00175) to ExStraCS 2.0 without EK Mutation but now we have also applied TuRF (see Table 12). The application of TuRF to the generation of EK scores (while more time consuming) is likely improve the quality of scores, and in combination with ExSTraCS, improve overall performance. This choice was based on the observation that on larger datasets, MultiSURF was failing to rank modeled predictive attributes with some of the highest scores. In other words, we observed that in datasets with greater than 200 attributes the trade-off between EK scoring performance and computational time makes the application of TuRF preferable. Referring to Table 12, notice that even with TuRF ExSTraCS 1.0 completely failed to scale up to 2000 attributes after 200,000 iterations evidenced by a testing accuracy close to 0.5 (no better than a random prediction), and all power scores of zero. The incorporation of RSL alone (No EK Mut) in ExS-TraCS 2.0 yielded only a small significant increase in testing accuracy and rule generality, along with a non-significant increases in ‘Single Power’, a marginally significant increase in ‘Single Co-Power’, and a significant reduction in run time.

With the addition of EK mutation, we observe a significant and dramatic increase in testing accuracy, ‘Single Power’, and ‘Single Co-Power’ and another significant reduction in run time (see Table 13. While ExS-TraCS 2.0 is clearly now able to learn at least parts of the underlying pattern, we still do not observe increases in the conservative ‘Both Power’ or ‘Both Co-Power’ metrics within 200,000 learning iterations.

As a follow up, we repeated our evaluation of ExS-TraCS 2.0 with EK mutation, but doubled the run parameter N from 2000 to 4000, to see if an increased population size might facilitate learning. These results are included in Table 14 under N = 4000. This change alone did little to improve learning if not hinder prediction performance, as the testing accuracy dropped by a small non-significant amount and both macro population size and run time increased dramatically. This suggests that N has already been set large enough to allow for the underlying relationships between attributes and class endpoint to be learned.

As further follow up we ran ExSTraCS 2.0 with EK mutation for twice as many iterations (i.e. 400,000 iterations) using both N = 2000 and N = 4000. These results with N = 2000 are summarized in Table 15. We can see that running ExSTraCS 2.0 for twice as many iterations continues to improve testing accuracy and the power metrics by a small non-significant amount. Additionally we observe that additional learning iterations significantly increases rule generality, and dramatically reduces macro population size as ExSTraCS seeks to further generalize accurate rules.

Lastly, we illustrate the importance of TuRF, and associated EK scores in helping to guide ExSTraCS in scaled-up analyses. Table 16 compares ExSTraCS 2.0 using all of the best setting explored thus far, with and without the application of TuRF. At 2000 attributes, TuRF and the associated higher-quality EK scores in this noisy problem environment is a critical part of the ExSTraCS 2.0 success.

These results suggest that ExSTraCS 2.0 with RSL and EK mutation has the ability to scale up to datasets with at least 100 times as many attributes than Michigan LCS have previously been applied in this type of complex, noisy problem domain. However, in these larger-scale, complex domains, the algorithm may require a greater number of learning iterations in order to be useful not only as a prediction machine, but as a source for knowledge discovery. While increasing testing accuracies provide the clearest indicator that ExSTraCS 2.0 is successfully scaling up to handle more attributes, knowledge discovery is equally important in our epidemiological problem domain of interest. Following rule inspection of some of the best performing [P]'s, we believe that at least one of the issues hindering our ability to achieve better power performance, is similarly tied to the fitness issue discussed earlier with regards to 'Both Co-Power'. Consider that as the number of attributes becomes larger, there will be an increasing probability that random combinations of attributes specified in rules will have high rule accuracies by chance alone (even in the context of using RSL). These non-optimal rules become more frequent in [P] as the algorithm learns, obscuring the identification of attributes that are most important to making optimal generalizations (i.e. that have an optimal balance of accuracy and coverage of instances in the dataset). Ultimately we return to the issue of how fitness is calculated, which conveys a localized perspective of rule-value when it relies on accuracy (where accuracy is the number of times the rule has made a correct prediction, only among the instances it has actually matched). We believe this to be a critical target for future investigation that may further expand the scalability and overall performance of ExS-TraCS and other Michigan LCS algorithms, especially in the context of complex, noisy, supervised learning problems.

Our final simulation study examines the various multiplexer problems. Figure 7 summarizes the results for the 6-bit, 11-bit, 20-bit, and 37-bit multiplexer problems when applying ExSTraCS 2.0 with EK Mutation and MultiSURF (no TuRF). Included in the figure are curves illustrating estimated training accuracy calculated every 1000 iterations based on the proceeding 1000 iterations. Also included are associated points which represent complete batch-wise testing accuracies representing predictive performance. Any space between a point and it's corresponding training curve represents overfitting by ExSTraCS. To best compare these results to other LCS investigations of multiplexer problems, readers should focus on these testing points. All curves and points are the average of 30 separate runs as described in [23]. We consider a multiplexer problem to have been solved when the testing accuracy goes above .99. Notice in Figure 7 that ExSTraCS 2.0 (1) solves the 6-bit and 11-bit problems within 10,000 iterations, (2) solves the 20-bit problem within 50,000 iterations with a training set of 10,000, 5,000 or 2,000, but has nearly solved it (test accuracy = 0.988) after only 10,000 iterations with a training set of 10,000, and (3) solves the 37-bit problem within 50,000 iterations with a training set of 10,000 or 5,000. As would be expected, larger training sets yield more efficient learning performance. Compared to the best results presented in [23] where either XCS or XCSCFC performed best, ExSTraCS 2.0 (1) yields very similar performance in solving the 6-bit and 11-bit multiplexer, (2) solves the 20-bit multiplexer in approximately 10,000 fewer training iterations, and (3) solves the 37-bit problem in approximately 20,000 fewer learning iterations. While ExSTraCS had access to all possible mulitplexer training instances in both the 6-bit and 11-bit problems, only a finite subset were available for larger problems. For example in the 20-bit multiplexer problem ExSTraCS only had access to a random set of approximately 0.95%, 0.48%, or 0.19% of all possible training instances. Additionally, compared to XCSCFC [23] ExSTraCS learns directly on a single given dataset as opposed to reusing building blocks from simpler multiplexer problems (e.g. using building blocks from solving the 20-bit problem to more efficiently solve the 37-bit problem). Therefore, we claim that ExSTraCS 2.0 can solve these problems directly in fewer iterations, with fewer unique training instances than previously reported in Michigan-style LCSs.

Figure 8 summarizes the results for the 70-bit and 135-bit multiplexer problems again applying ExSTraCS 2.0. In this Figure, the 70-bit problem is run up to 500,000 learning iterations, while the 135-bit problem is run up to 1,500,000 iterations. Notice that ExSTraCS 2.0 (1) solves the 70-bit problem within 200,000 iterations with a training set of 20,000 and 10,000, and (2) solves the 135-bit problem within 1,500,000 iterations with a training set of 40,000. Again compared to the best results presented in [23], ExSTraCS 2.0 (1) solves the 70-bit problem in approximately 200,000 fewer learning iterations, and (2) has nearly solved the 135-bit multiplexer problem (test accuracy = 0.98) in approximately 300,000 fewer iterations. We also consider the results reported for MPLCS [4], a powerful parallelized Pittsburgh-style LCS, that reported solving the 70-bit multiplexer problem. While the results of this system are difficult to directly compare to ExSTraCS, the authors determined their algorithm to require a little over 1.1 million learning iterations (put into the context of a Michigan-style LCS). While ExSTraCS outperforms MPLCS using learning iterations as a performance metric, MPLCS boasts a very rapid run time attributed to its efficient coding, windowing scheme, and parallelized computation [4]. ExSTraCS 2.0 is currently implemented in Python, and has thus far been designed primarily for learning efficacy rather than efficiency and runs significantly slower than MPLCS or Java implementations of XCS. For example, ExSTraCS 2.0 took approximately 7 hours to solve the 70-bit problem. We fully anticipate dramatic significant reductions in run time once we have the opportunity to implement ExS-TraCS in a faster coding language, optimize code efficiency, and incorporate parallelization strategies. While we acknowledge run-time to be a shortcoming of the the current ExSTraCS implementations, we believe that a fair run-time comparison to other cutting-edge implementation can only be made after we have had the opportunity to address these accessible issues.

4 Conclusions

This work has focused on the scalability of Michigan LCS algorithms. In particular, we have expanded the ExSTraCS algorithm, which was specifically designed to address classification, prediction, and knowledge discovery in complex, noisy, supervised learning problems. In this paper we have (1) detailed the ExSTraCS 2.0 algorithm, (2) identified the advantages of applying a Michigan LCS as a machine learning algorithm, (3) developed and evaluated a data-driven rule specificity limit (RSL) and an expert knowledge (EK) guided mutation operator for rule discovery in data with a greater number of potentially predictive attributes, (4) demonstrated that ExSTraCS 2.0 dramatically improves performance over a spectrum of small scale complex simulated datasets (5) added the TuRF wrapper algorithm to ExSTraCS to improve the quality of EK score discovery, (6) demonstrated that ExSTraCS 2.0 can functionally scale up to datasets with at least 2000 attributes, (7) demonstrated that ExSTraCS 2.0 can solve all multiplexer problems up to and including the 135-bit multiplexer (8) eliminated the need for multiple run parameters, effectively facilitating use of the algorithm, and (9) identified key targets for the ongoing development of a Michigan LCS algorithm that can more effectively scale up to datasets with as many attributes as may be computationally feasible.

This work re-emphasizes the importance of applying available EK to guide machine learning in complex and larger-scale domains. Without EK incorporated into mutation, ExSTraCS was unable to scale up performance to datasets with 2000 attributes. Notably, the quality of EK weights is just as important to scalability as their application to the algorithm.

Based on trends observed in this study, we expect that ExSTraCS 2.0 will perform best and allow maximum scalability when (1) a greater number of training instances are available, (2) there is less noise obscuring the underlying pattern of association, (3) ExSTraCS is allowed to learn for a greater number of training iterations, (4) TuRF is used in combination with an EK generation algorithm, and (5) if possible, some reliable feature selection can be applied to pre-process the dataset and limit the number of attributes that will be presented to ExSTraCS. Expanding on this last item, the ExSTraCS algorithm itself could be applied to perform feature selection. Previously, we applied AF-UCS (a precursor to ExSTraCS) in an epidemiological investigation of bladder cancer [40]. In this analysis, AF-UCS was run once to identify significantly over-specified attributes as described in [42], and then again on the subset of the data that included these top attributes. This yielded a more accurate and interpretable rule population.

In future work, we believe that the following areas would be particularly beneficial to address in ExSTraCS and other Michigan LCSs: (1) a new strategy for the calculation of rule fitness in supervised Michigan LCSs, (2) a more intuitive deletion scheme for noisy problem domains, (3) a strategy to adapt the algorithm to learn on continuous endpoints, and (4) exploring parallelization and ensemble learning to scale up to truly large-scale datasets i.e. on the order of hundreds of thousands to millions of attributes.

Acknowledgments

This work was supported by NIH grants AI59694, LM009012, LM010098, EY022300, LM011360, CA134286, and GM103534.

References

1. Bacardit J, Burke E, Krasnogor N. Improving the scalability of rule-based evolutionary learning. Memetic Computing. 2009; 1(1):55–67.
2. Bacardit, J.; Goldberg, D.; Butz, M.; Llora, X.; Garrell, J. Parallel Problem Solving from Nature-PPSN VIII. Springer; 2004. Speeding-up pittsburgh lcss: Modeling time and accuracy; p. 1021-1031.
3. Bacardit, J.; Krasnogor, N. A mixed discrete-continuous attribute list representation for large scale classification domains. Proceedings of the 11th Annual conference on Genetic and evolutionary computation; ACM; 2009. p. 1155-1162.
4. Bacardit J, Krasnogor N. Performance and efficiency of memetic pittsburgh learning classifier systems. Evolutionary computation. 2009; 17(3):307–342. [PubMed: 19708771]
5. Bacardit J, Llorà X. Large-scale data mining using genetics-based machine learning. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2013; 3(1):37–61.
6. Bacardit, J.; Stout, M.; Hirst, J.; Sastry, K.; Llorà, X.; Krasnogor, N. Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. Proceedings of the 9th annual conference on Genetic and evolutionary computation; ACM; 2007. p. 346-353.
7. Bagallo G, Haussler D. Boolean feature discovery in empirical learning. Machine learning. 1990; 5(1):71–99.
8. Barto, AG.; Anandan, P.; Anderson, CW. Adaptive and Learning Systems. Springer; 1986. Cooperativity in networks of pattern recognizing stochastic learning automata; p. 235-246.
9. Bernadó-Mansilla E, Garrell-Guiu J. Accuracy-based learning classifier system: models, analysis and applications to classification tasks. Evolutionary Computation. 2003; 11(3):209–238. [PubMed: 14558911]
10. Bull L, Studley M, Bagnall A, Whittley I. Learning classifier system ensembles with rule-sharing. Evolutionary Computation, IEEE Transactions on. 2007; 11(4):496–502.
11. Butz, MV. Rule-Based Evolutionary Online Learning Systems. Springer; 2006. Xcs in binary classification problems; p. 147-156.
12. Butz MV, Goldberg DE, Tharakunnel K. Analysis and improvement of fitness exploitation in xcs: Bounding models, tournament selection, and bilateral accuracy. Evolutionary Computation. 2003; 11(3):239–277. [PubMed: 14558912]
13. Butz MV, Kovacs T, Lanzi PL, Wilson SW. Toward a theory of generalization and learning in xcs. Evolutionary Computation, IEEE Transactions on. 2004; 8(1):28–46.
14. Butz, MV.; Pelikan, M. Studying xcs/boa learning in boolean functions: structure encoding and random boolean functions. Proceedings of the 8th annual conference on Genetic and evolutionary computation; ACM; 2006. p. 1449-1456.
15. Butz, MV.; Sastry, K.; Goldberg, DE. Genetic and Evolutionary ComputationGECCO 2003. Springer; 2003. Tournament selection: Stable fitness pressure in xcs; p. 1857-1869.
16. DeJong, KA.; Spears, WM. Tech rep, DTIC Document. 1990. Learning concept classification rules using genetic algorithms.
17. Franco, M.; Krasnogor, N.; Bacardit, J. Speeding up the evaluation of evolutionary learning systems using gpg-pus. Proceedings of the 12th annual conference on Genetic and evolutionary computation; ACM; 2010. p. 1039-1046.
18. Granizo-Mackenzie, D.; Moore, JH. Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Springer; 2013. Multiple threshold spatially uniform relieff for the genetic analysis of complex human diseases; p. 1-10.
19. Greene C, Penrod N, Kiralis J, Moore J. Spatially uniform relieff (surf) for computationally-efficient filtering of gene-gene interactions. BioData mining. 2009; 2(1):1–9. [PubMed: 19216798]
20. Greene, CS.; Himmelstein, DS.; Kiralis, J.; Moore, JH. Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Springer; 2010. The informative extremes: using both nearest and farthest individuals can improve relief algorithms in the domain of human genetics; p. 182-193.

21. Higuchi, T.; Niwa, T.; Tanaka, T.; Iba, H.; de Garis, H.; Furuya, T. Evolvable hardware–genetic-based generation of electric circuitry at gate and hardware description language (hdl) levels. Electrotechnical Laboratory; Tsukuba, Japan: 1993. p. 93-4.

22. Iba H, De Garis H, Sato T. Genetic programming using a minimum description length principle. Advances in genetic programming. 1994; 1:265–284.

23. Iqbal, M.; Browne, WN.; Zhang, M. Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. 2012.

24. Iqbal, M.; Browne, WN.; Zhang, M. Extending learning classifier system with cyclic graphs for scalability on complex, large-scale boolean problems. Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference; ACM; 2013. p. 1045-1052.

25. Kononenko, I. Machine Learning: ECML-94. Springer; 1994. Estimating attributes: analysis and extensions of relief; p. 171-182.

26. Lanzi, PL.; Loiacono, D. Learning Classifier Systems. Springer; 2010. Speeding up matching in learning classifier systems using cuda; p. 1-20.

27. Llorà, X.; Sastry, K. Fast rule matching for lcss via vector instructions. Proceedings of the 8th annual conference on Genetic and evolutionary computation; ACM; 2006. p. 1513-1520.

28. Moore J, White B. Tuning relieff for genome-wide genetic analysis. Evolutionary computation, machine learning and data mining in bioinformatics. 2007:166–175.

29. Moore, JH. Epistasis. Springer; 2015. Epistasis analysis using relieff; p. 315-325.

30. Moore JH, Asselbergs FW, Williams SM. Bioinformatics challenges for genome-wide association studies. Bioinformatics. 2010; 26(4):445–455. [PubMed: 20053841]

31. Moore JH, Ritchie MD. The challenges of whole-genome approaches to common diseases. Jama. 2004; 291(13):1642–1643. [PubMed: 15069055]

32. Quinlan, JR. ML. 1988. An empirical comparison of genetic and decision-tree classifiers; p. 135-141.

33. Rudd J, Moore JH, Urbanowicz RJ. A multi-core parallelization strategy for statistical significance testing in learning classifier systems. Evolutionary intelligence. 2013; 6(2):127–134.

34. Smith RE, Cribbs HB III. Is a learning classifier system a type of neural network? Evolutionary Computation. 1994; 2(1):19–36.

35. Tan J, Moore J, Urbanowicz R. Rapid rule compaction strategies for global knowledge discovery in a supervised learning classifier system. Advances in Artificial Life, ECAL. 2013; 12:110–117.

36. Urbanowicz, R. [accessed August 24, 2014] ExSTraCS 2.0. 2014. URL http://http://sourceforge.net/projects/exstracs/

37. Urbanowicz, R.; Granizo-Mackenzie, A.; Moore, J. Instance-linked attribute tracking and feedback for michigan-style supervised learning classifier systems. Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference; ACM; 2012. p. 927-934.

38. Urbanowicz, R.; Moore, J. The application of michigan-style learning classifier systems to address genetic heterogeneity and epistasis in association studies. Proceedings of the 12th annual conference on Genetic and evolutionary computation; ACM; 2010. p. 195-202.

39. Urbanowicz, R.; Moore, J. Parallel Problem Solving from Nature–PPSN XI. 2011. The application of pittsburgh-style lcs to address genetic heterogeneity and epistasis in association studies; p. 404-413.

40. Urbanowicz RJ, Andrew AS, Karagas MR, Moore JH. Role of genetic heterogeneity and epistasis in bladder cancer susceptibility and outcome: a learning classifier system approach. Journal of the American Medical Informatics Association. 2013

41. Urbanowicz, RJ.; Bertasius, G.; Moore, JH. Parallel Problem Solving from Nature-PPSN XIII. Springer; 2014. An extended michigan-style learning classifier system for flexible supervised learning, classification, and data mining. in press

42. Urbanowicz RJ, Granizo-Mackenzie A, Moore JH. An analysis pipeline with statistical and visualization-guided knowledge discovery for michigan-style learning classifier systems. Computational Intelligence Magazine, IEEE. 2012; 7(4):35–45.

43. Urbanowicz, RJ.; Granizo-Mackenzie, D.; Moore, JH. Parallel Problem Solving from Nature-PPSN XII. Springer; 2012. Using expert knowledge to guide covering and mutation in a michigan style learning classifier system to detect epistasis and heterogeneity; p. 266-275.

44. Urbanowicz RJ, Kiralis J, Fisher JM, Moore JH. Predicting the difficulty of pure, strict, epistatic models: metrics for simulated model selection. BioData mining. 2012; 5(1):1–13. [PubMed: 22297131]

45. Urbanowicz RJ, Kiralis J, Sinnott-Armstrong NA, Heberling T, Fisher JM, Moore JH. Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. BioData mining. 2012; 5(1):16. [PubMed: 23025260]

46. Urbanowicz RJ, Moore JH. Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications. 2009

47. Velez DR, White BC, Motsinger AA, Bush WS, Ritchie MD, Williams SM, Moore JH. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. Genetic epidemiology. 2007; 31(4):306–315. [PubMed: 17323372]

48. Wilson S. Classifier fitness based on accuracy. Evolutionary computation. 1995; 3(2):149–175.

49. Wilson SW. Classifier systems and the animat problem. Machine learning. 1987; 2(3):199–228.

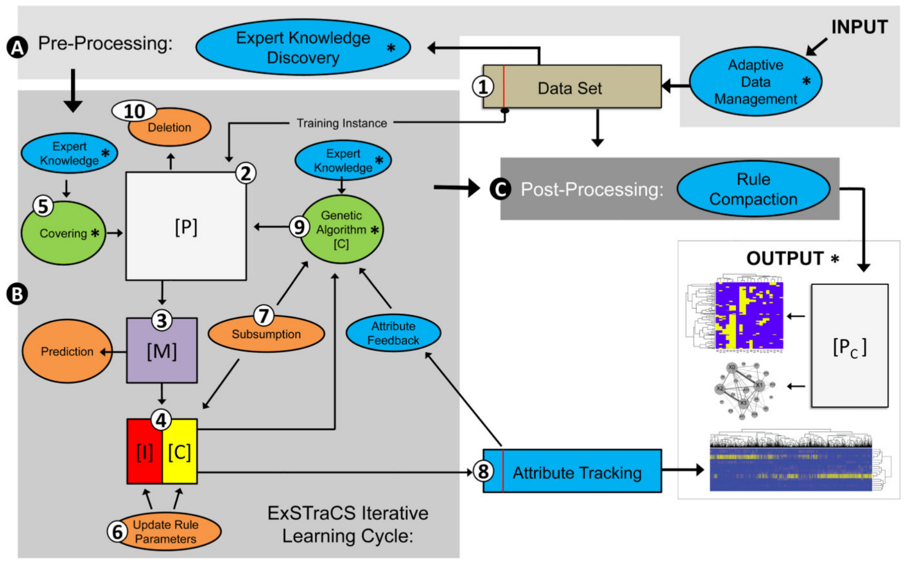

Fig. 1. ExSTraCS Schematic

Ovals are mechanisms, bordered squares are sets of either data or classifiers, green items are classifier discovery mechanisms, orange items are traditional LCS mechanisms, blue items are mechanisms unique to ExSTraCS, and items with a (*) have features new to ExSTraCS 2.0.

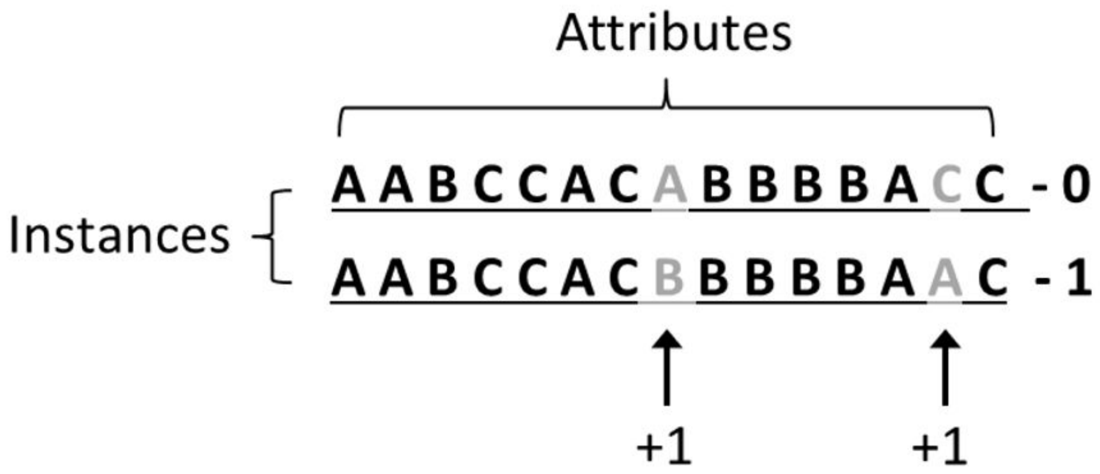
Attributes
Instances
A A B C C A C A B B B B A C C - 0
A A B C C A C B B B B B A A C - 1
+1
+1

Fig. 2.
Basics of Relief-based algorithms.

Quaternary Knowledge Representation

Rule Condition:	[#, 2, #, #, 0, #, 1, 2, #, #]
Classification/Action:	1

Mixed Discrete-Continuous Attribute-List Knowledge Representation

Attribute Reference:	[1, 4, 6, 7]
Rule Condition:	[2, 0, [0.4 - 0.7], 'high']
Classification/Action:	1

Fig. 3. Knowledge Representations

Quaternary vs. Mixed Discrete-Continuous Attribute List. A continuous-valued attribute is highlighted in grey.

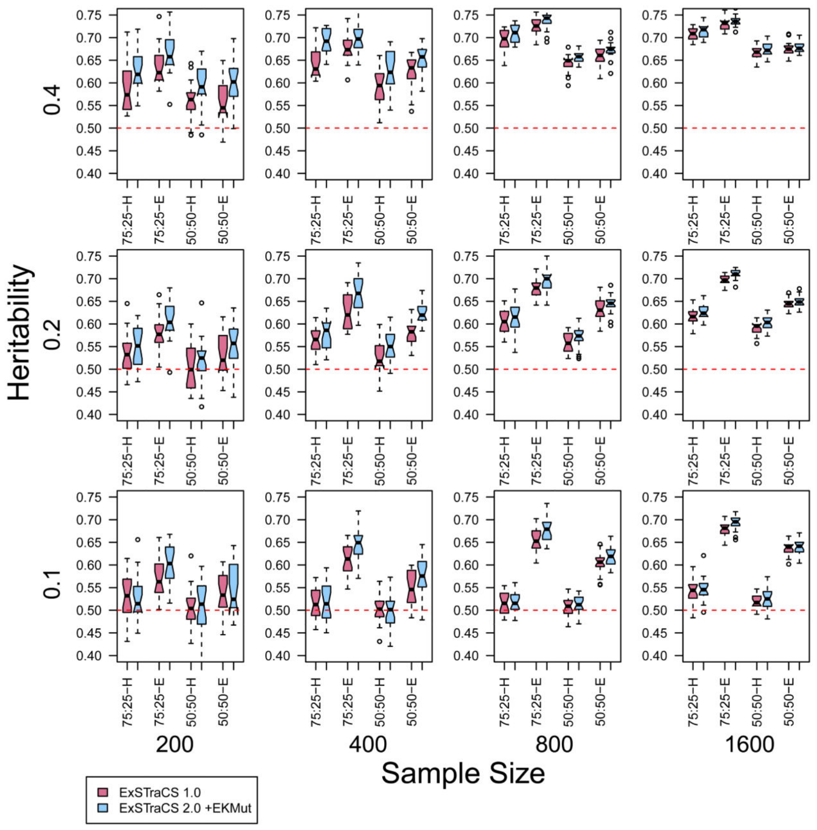

Fig. 4.
Comparing Testing Accuracy after 200,000 iterations over all dataset combinations in the simulation study.

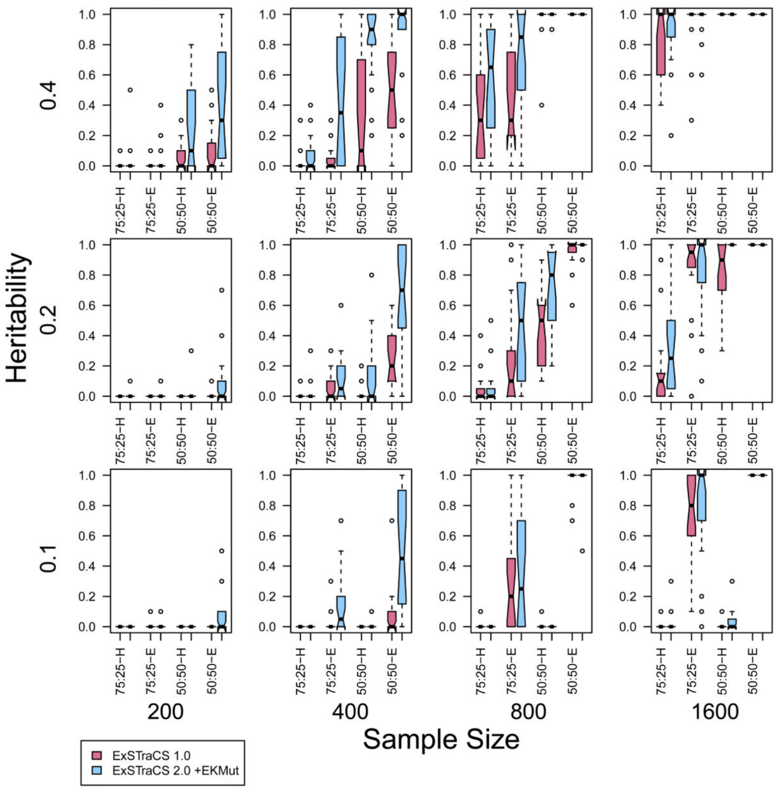

Fig. 5.
Comparing ‘Both Power’ after 200,000 iterations over all dataset combinations in the simulation study.

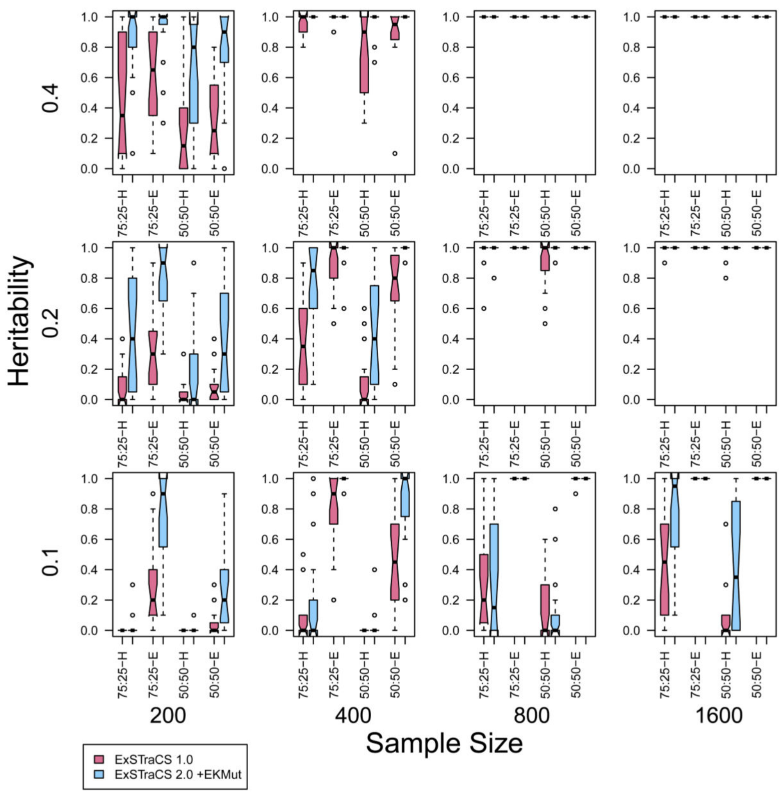

Fig. 6.

Comparing ‘Single Power’ after 200,000 iterations over all dataset combinations in the simulation study.

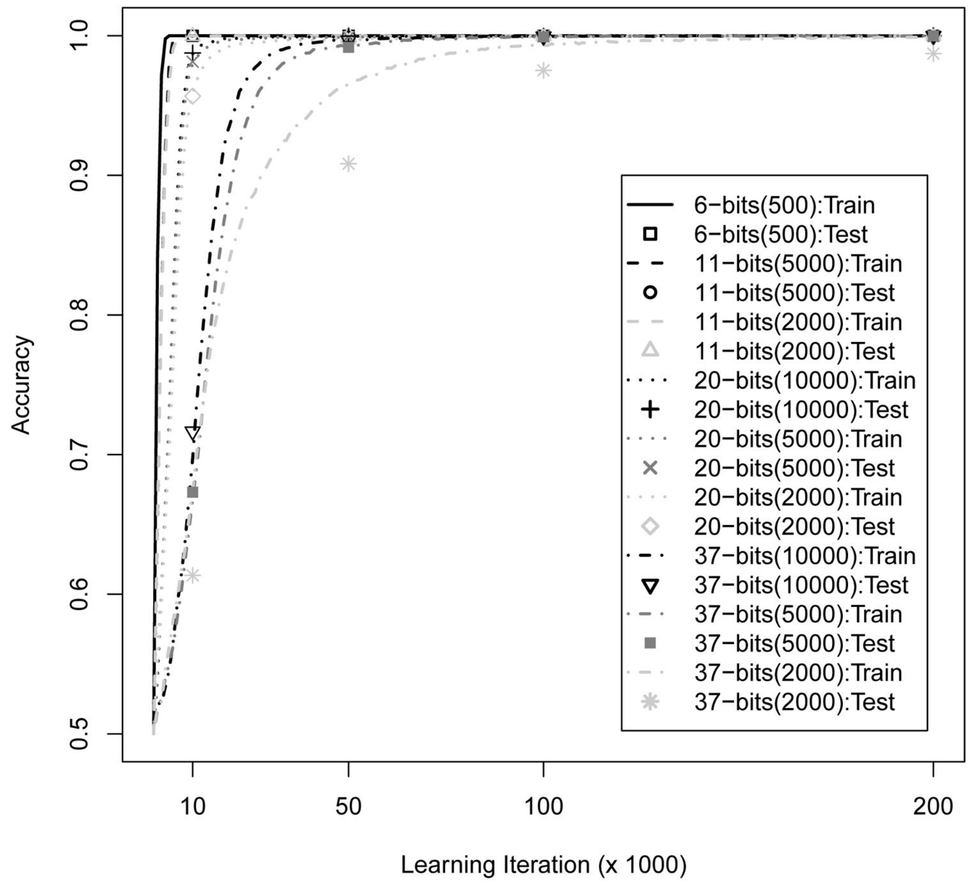

Fig. 7.

Multiplexer Performance (6-bit, 11-bit, 20-bit and 37-bit). The values given in parenthesis are respective training/testing dataset sizes. Lines illustrate training accuracy estimations calculated over the last 1000 iterations. Points illustrate complete batch-wise testing accuracies (i.e. predictive performance) calculated over a corresponding testing dataset.

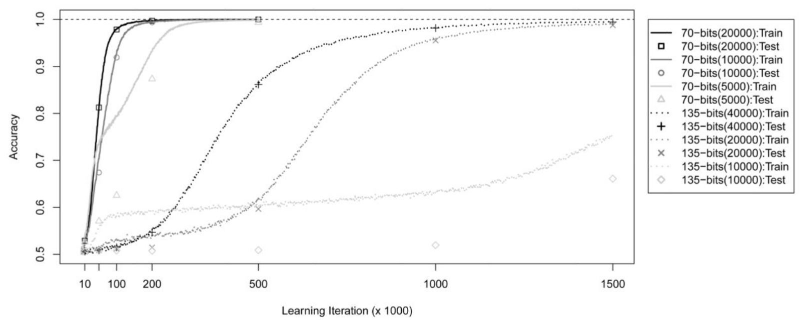

Fig. 8.

Multiplexer Performance (70-bit and 135-bit). The values given in parenthesis are respective training/testing dataset sizes. Lines illustrate training accuracy estimations calculated over the last 1000 iterations. Points illustrate complete batch-wise testing accuracies (i.e. predictive performance) calculated over a corresponding testing dataset. The dashed line illustrates perfect accuracy.

Table 1

Example calculations of ψ

	ε			
n	**2**	**3**	**4**	**5**
1	2	3	4	5
2	4	9	16	25
3	8	27	64	125
4	16	81	256	625
5	32	243	1024	3125
6	64	729	4096	15625
7	128	2187	16384	78125
8	256	6561	65536	390625

Table 2

ExSTraCS Classifier Parameters

Parameter	Description	Use	Updated
MatchCount	Number of times included in [M]	Calculate accuracy	When in [M]
CorrectCount	Number of times included in [C]	Calculate accuracy	When in [C]
Accuracy	CorrectCount/MatchCount	Calculate fitness, subsumption activation	When in [M]
Fitness	Estimated health/value of classifier	Selection, deletion, prediction	When in [M]
Numerosity	Number of classifier copies in [P]	Used throughout	GA or subsumption
AveMatchSetSize	Average size of match sets this classifier has been a part of	Deletion	When in [M]
TimeStampGA	The most recent iteration that this classifier was in a [C]	Activating GA	When in [C]
InitTimeStamp	The learning iteration when the classifier was first generated	Epoch Status	Never
DeletionVote	A calculated weight for selecting this classifier for deletion	Deletion	Whenever deletion is required

Table 3

ExSTraCS Run Parameters

Parameter	**Default Setting**	**Description**
I_{max}	200,000	Maximum Learning Iterations
N	2000	Maximum Population Size
ν	1	Calculation of fitness from accuracy
χ	0.8	Probability of crossover
υ	0.04	A new mutation pressure
θ_{GA}	25	GA activation threshold
θ_{del}	20	Protective deletion threshold
θ_{sub}	20	Subsumption activation threshold
acc_{sub}	0.99	Subsumption accuracy threshold
β	0.2	Update of AveMatchSetSize
δ	0.1	Deletion weight calculation threshold
$init_{fit}$	0.01	Initial fitness for new rules
fitnessReduction	0.1	Penalize fitness of offspring rule following crossover
$theta_{sel}$	0.5	Tournament selection proportion
EK_Algorithm	MultiSURF	Specifies an internal source of EK generation
TuRF_Percent	0.2	Percent of attributes removed each score re-estimation

Table 4

Characteristics of *x*-multiplexer problems.

x	Address Bits	Order of Interaction	Heterogeneous Combinations	Unique Instances	Optimal Rules [O]
6-bit	2	3	4	64	8
11-bit	3	4	8	2048	16
20-bit	4	5	16	1.05×10^{6}	32
37-bit	5	6	32	1.37×10^{11}	64
70-bit	6	7	64	1.18×10^{21}	128
135-bit	7	8	128	4.36×10^{40}	256

Table 5

Learning parameters for *x*-multiplexer problems.

x	N	I_{max}	Train/Test Data Size(s)
6-bit	500	200,000	500
11-bit	1000	200,000	2000, 5000
20-bit	2000	200,000	2000, 5000, 10000
37-bit	5000	200,000	2000, 5000, 10000
70-bit	10000	500,000	5000, 10000, 20000
135-bit	10000	1,500,000	10000, 20000, 40000

Table 6

20 Attribute Simulation Study: Influence of p_{spec} in ExSTraCS 1.0

200,000 Iterations +QRF			
Performance Statistics	**ExSTraCS 1.0**		
	$p_{spec} = 0.5$	$p_{spec} = 0.175$	p
Train Accuracy	.8566	.8564	-
Test Accuracy	.6001	.6004	-
Both Power	.3063	.3031	-
Single Power	.6167	.6115	-
Both Co-Power	.1927	.1990	-
Single Co-Power	.7604	.7552	-
Rule Generality	.7602	.7607	↑ ***
Macro Population	1049.7	1048.9	↓ *
Run Time (min)	48.68	47.04	↓ ***

– No significant change

* p < 0.05 (Direction of change given by arrows)

** p < 0.0025 applying less conservative Bonferroni correction

*** p < 0.00027 applying Bonferroni correction

NOTE: Tables 6 through 16 all use this key.

Table 7

20 Attribute Simulation Study: ExSTraCS 1.0 vs. 2.0 Influence of RSL

200,000 Iterations +QRF			
Performance Statistics	**ExSTraCS 1.0**	**ExSTraCS 2.0**	
	p_{spec} = 0.175	**No EK Mut**	***p***
Train Accuracy	.8564	.7999	↓ ***
Test Accuracy	.6004	.6070	↑ ***
Both Power	.3031	.3667	↑ ***
Single Power	.6115	.7146	↑ ***
Both Co-Power	.1990	.1667	↓ ***
Single Co-Power	.7552	.7792	↑ **
Rule Generality	.7607	.8025	↑ ***
Macro Population	1048.9	1085.12	↑ ***
Run Time (min)	47.04	58.55	↑ ***

Table 8

20 Attribute Simulation Study: ExSTraCS 2.0 Influence of EK Mutation

200,000 Iterations +QRF			
Performance Statistics	**ExSTraCS 2.0**		
	No EK Mut	**EK Mut**	***p***
Train Accuracy	.7999	.7882	↓ ***
Test Accuracy	.6070	.6178	↑ ***
Both Power	.3667	.4052	↑ ***
Single Power	.7146	.7635	↑ ***
Both Co-Power	.1667	.0438	↓ ***
Single Co-Power	.7792	.7771	-
Rule Generality	.8025	.8026	-
Macro Population	1085.12	987.12	↓ ***
Run Time (min)	58.55	54.77	↓ ***

Table 9

200 Attribute Simulation Study: ExSTraCS 1.0 Influence of TuRF

200,000 Iterations +QRF, p_{spec} = 0.0175			
Performance Statistics	**ExSTraCS 1.0**		
	MultiSURF	**+TuRF**	***p***
Train Accuracy	.7004	.7206	↑ ***
Test Accuracy	.6064	.6393	↑ ***
Both Power	0	0	-
Single Power	0	0	-
Both Co-Power	0	.015	-
Single Co-Power	.05	.70	↑ **
Rule Generality	.9415	.9482	↑ ***
Macro Population	903.84	980.87	↑ ***
Run Time (min)	65.47	62.15	↓ **

Table 10

200 Attribute Simulation Study: ExSTraCS 1.0 vs. 2.0 Influence of RSL

200,000 Iterations +QRF			
Performance Statistics	**ExSTraCS 1.0**	**ExSTraCS 2.0**	
	p_{spec} **= 0.0175**	**No EK Mut**	***p***
Train Accuracy	.7004	.7565	↑ ***
Test Accuracy	.6064	.6690	↑ ***
Both Power	0	1	↑ ***
Single Power	0	1	↑ ***
Both Co-Power	0	1	↑ ***
Single Co-Power	.05	1	↑ ***
Rule Generality	.9415	.9794	↑ ***
Macro Population	903.84	1169.36	↑ ***
Run Time (min)	65.47	60.11	↓ ***

Table 11

200 Attribute Simulation Study: ExSTraCS 2.0 Influence of EK Mutation

200,000 Iterations +QRF			
Performance Statistics	**ExSTraCS 2.0**		
	No EK Mut	**EK Mut**	***p***
Train Accuracy	.7565	.7555	-
Test Accuracy	.6690	.6734	*p* = 0.058
Both Power	1	1	-
Single Power	1	1	-
Both Co-Power	1	1	-
Single Co-Power	1	1	-
Rule Generality	.9794	.9798	↑ ***
Macro Population	1169.36	1120.27	↓ ***
Run Time (min)	60.11	60.11	-

Table 12

2000 Attribute Simulation Study: ExSTraCS 1.0 vs. 2.0 Influence of RSL

200,000 Iterations +QRF, +TuRF			
Performance Statistics	**ExSTraCS 1.0**	**ExSTraCS 2.0**	
	p_{spec} **= 0.00175**	**No EK Mut**	***p***
Train Accuracy	.6528	.7156	↑ ***
Test Accuracy	.5056	.5419	↑ **
Both Power	0	0	-
Single Power	0	0.15	-
Both Co-Power	0	0	-
Single Co-Power	0	0.35	↑ *
Rule Generality	.9688	.9978	↑ ***
Macro Population	791.185	1396.41	↑ ***
Run Time (min)	113.12	96.44	↓ ***

Table 13

2000 Attribute Simulation Study: ExSTraCS 2.0 Influence of EK Mutation

200,000 Iterations +QRF, +TuRF			
Performance Statistics	**ExSTraCS 2.0**		
	No EK Mut	**EK Mut**	***p***
Train Accuracy	.7156	.7198	-
Test Accuracy	.5419	.5959	↑ ***
Both Power	0	0	-
Single Power	0.15	0.85	↑ ***
Both Co-Power	0	0.1	-
Single Co-Power	0.35	1	↑ **
Rule Generality	.9978	.9978	-
Macro Population	1396.41	1411.83	↑ ***
Run Time (min)	96.44	91.65	↓ **

Table 14

2000 Attribute Simulation Study: ExSTraCS 2.0 Influence *N*

200,000 Iterations +QRF, +TuRF, +EK Mutation			
Performance Statistics	**ExSTraCS 2.0**		
	***N* = 2000**	***N* = 4000**	***p***
Train Accuracy	.7198	.7555	↑ ***
Test Accuracy	.5959	.5916	-
Both Power	0	0.1	-
Single Power	0.85	0.9	-
Both Co-Power	0.1	0.25	-
Single Co-Power	1	1	-
Rule Generality	.9978	.9977	↓ ***
Macro Population	1411.83	2802.91	↑ ***
Run Time (min)	91.65	184.18	↑ ***

Table 15

2000 Attribute Simulation Study: ExSTraCS 2.0 Additional Learning Iterations (i.e. 200,000 vs. 400,000)

***N* = 2000, +QRF, +TuRF, +EK Mutation**			
Performance Statistics	**ExSTraCS 2.0**		
	200,000	**400,000**	***p***
Train Accuracy	.7198	.7228	-
Test Accuracy	.5959	.5960	-
Both Power	0	0.1	-
Single Power	0.85	1	-
Both Co-Power	0.1	0.2	-
Single Co-Power	1	1	-
Rule Generality	.9978	.9981	↑ ***
Macro Population	1411.83	1203.09	↓ ***
Run Time (min)	91.65	169.39	↑ ***

Table 16

2000 Attribute Simulation Study: ExSTraCS 2.0 Influence of TuRF

400,000 Iterations, *N* = 2000, +QRF, +EK Mutation			
Performance Statistics	**ExSTraCS 2.0**		
	MultiSURF	**+TuRF**	***p***
Train Accuracy	.6936	.7228	↑ ***
Test Accuracy	.5225	.5960	↑ ***
Both Power	0	0.1	-
Single Power	0	1	↑ ***
Both Co-Power	0	0.2	-
Single Co-Power	0.25	1	↑ ***
Rule Generality	.9981	.9981	-
Macro Population	1297.39	1203.09	↓ ***
Run Time (min)	158.44	169.39	↑ **

Algorithm 1

ExSTraCS 2.0 Covering

```
Require: inst
  $A_{Spec} \leftarrow$ random(1, $RSL$)
  if useEK then
    $i = 0$
    while $i < A\_Spec$ do
      targetAtt = EK_RankedAttList[i]
      Add targetAtt to attReference
      if targetAtt is discrete then
        Add inst[targetAtt] to ruleCondition
      else
        Add newRange(inst[targetAtt]) to ruleCondition
      end if
    end while
  else
    $i = 0$
    while $i < A_{Spec}$ do
      targetAtt =random(Att)
      Add targetAtt to attReference
      if targetAtt is discrete then
        Add newRange(inst[targetAtt]) to ruleCondition
      end if
    end while
  end if
  ruleClass ← instanceClass
  return rule
```