APPENDIX A
IMPLEMENTATION DETAILS AND SOFTWARE AVAILABILITY

A.1 FG-EA Details

The internal details of FG-EA are summarized in the diagram in Fig. 1. The population size, `GenSize`, set to N for generation 0, decreases by 10% over the previous generation. The decrease aims to address the aging problem observed in GP. Regularly decreasing population size over generations introduces selection pressure in a population and avoids meaningless computations in later generations [6].

The first population, generation 0, consists of $N = 15,00$ trees (features) generated at random. Features in subsequent generations are evolved by applying mutation and crossover over selected parent features. The parent features are obtained from a hall of fame, which keeps track of the fittest features in each generation. A surrogate fitness function estimates the fitness of each feature in a generation. The top ℓ features of a generation are copied over to a growing hall of fame. Then, m features are selected at random from the hall of fame to serve as parents for the next generation.

The process of evolving features continues for `NrGens` generations. This parameter is set to 25 in our application on the DNA splice site prediction problem. This upper bound is sufficient, as our results show convergence of fitness values after 20 generations.

A.1.1 Generating Random Initial Features

Generation 0 consists of $N = 15,000$ features, a choice is warranted by the complexity of the feature space. The trees representing features are generated using the well-known *ramped half-and-half* generative method [5]. The ramped half-and-half method incorporates both the Full and Grow techniques in order to obtain a mixture of fully-balanced trees and bushy trees with each technique is employed with equal probability of 0.5 [5]. A maximum initial tree depth D is set a priori to a small value (5 in our case, as proposed in [5] and employed by many GP algorithms [6]). Terminal nodes include characters from the DNA alphabet and sequence positions (as described in the paper).

Closure

GP relies on the principle of closure, which specifies that all trees generated are both syntactically and semantically correct. FG-EA employs strongly typed GP (STGP) in order to place additional type constraints on the nodes and specify which nodes may link with other nodes [8]. STGP allows the mutation and crossover operators to generate syntactically and semantically correct trees.

A.1.2 Genetic Operators

Given a set of m features extracted from the hall of fame to serve as parents in a generation, the rest of `GenSize`$- m$ features are generated using the mutation and crossover operators. Three breeding pipelines are employed, mutation, mutation-ERC, and crossover, as illustrated in the GP diagram in Fig. 1. Two types of mutations are possible in S-expressions, one that replaces an entire subtree, and another that replaces only ERCs. These three pipelines are executed in parallel in order to obtain new offspring until the goal population size is reached. As the mutation and mutation-ERC pipelines run, they each have a P_m probability of performing a mutation on a selected parent. As the crossover pipeline runs, it has a P_c probability of carrying out a crossover on a selected parent. In our implementation, $P_m = 0.5$ and $P_c = 0.9$. These are standard values in GP implementations [5]. As these pipelines are independent of one another, the probabilities do not have to sum to 1.

Mutation-ERC

A parent is selected using tournaments of size 7. With probability 0.1, a terminal node is selected in its tree. A non-terminal node is selected with probability 0.9. If the selected node is a terminal, hence an ERC, a new ERC is generated according to a Gaussian probability over the range of values of the ERC selected for replacement. If the selected node is a non-terminal, only the ERCs of the subtree rooted at the node are replaced. New values are generated for each ERC as above.

Mutation

After a node is selected for replacement, the mutation pipeline proceeds differently from the mutation-ERC pipeline. When the selected node is a terminal, the node is replaced with another ERC as in the mutation-ERC pipeline. Otherwise, the selected node is replaced with a semantically-correct tree with the same return type as the subtree chosen for replacement. Such a tree is generated using the standard GP Grow technique under the maximum depth restriction (17 in our experiments). The Grow technique is attempted a maximum of 5 times in order to obtain a tree that satisfies the return type constraint. This ensures that closure is maintained.

Crossover

The tournament selection scheme is carried out twice to obtain two individuals from a population. Once two parents are selected, their crossover proceeds as in the standard Koza-style *subtree crossover* [5]. Two subtrees, one from each parent, are selected at random. A random node is then chosen in each subtree. If the selected nodes have the same return type, and swapping the subtrees rooted at these nodes will not violate the maximum depth constraint, then the swap is performed. Otherwise, the process begins anew for a maximum of 5 times. While the crossover can result in two individuals, only one is maintained in FG-EA.

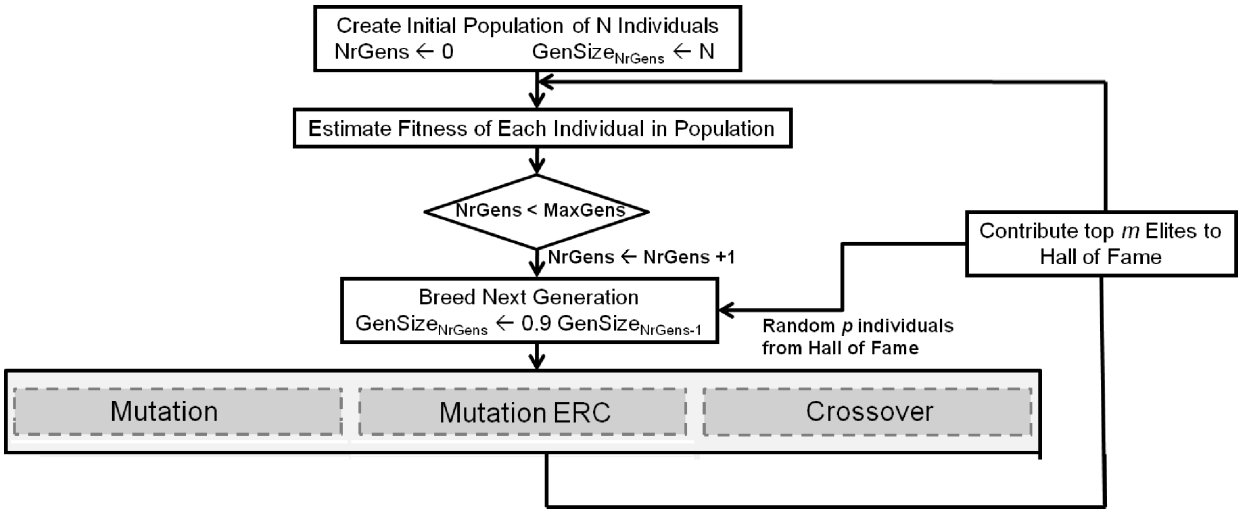

Fig. 1. The diagram summarizes the main steps in our FG-EA algorithm. Features/individuals are evolved until a maximum number Gen_Max of generations has been reached. The mutation and crossover operators detailed below are employed to obtain new features in a generation. Top features of a generation are contributed to a growing hall of fame which then in turn contributes randomly selected features to seed the next generation.

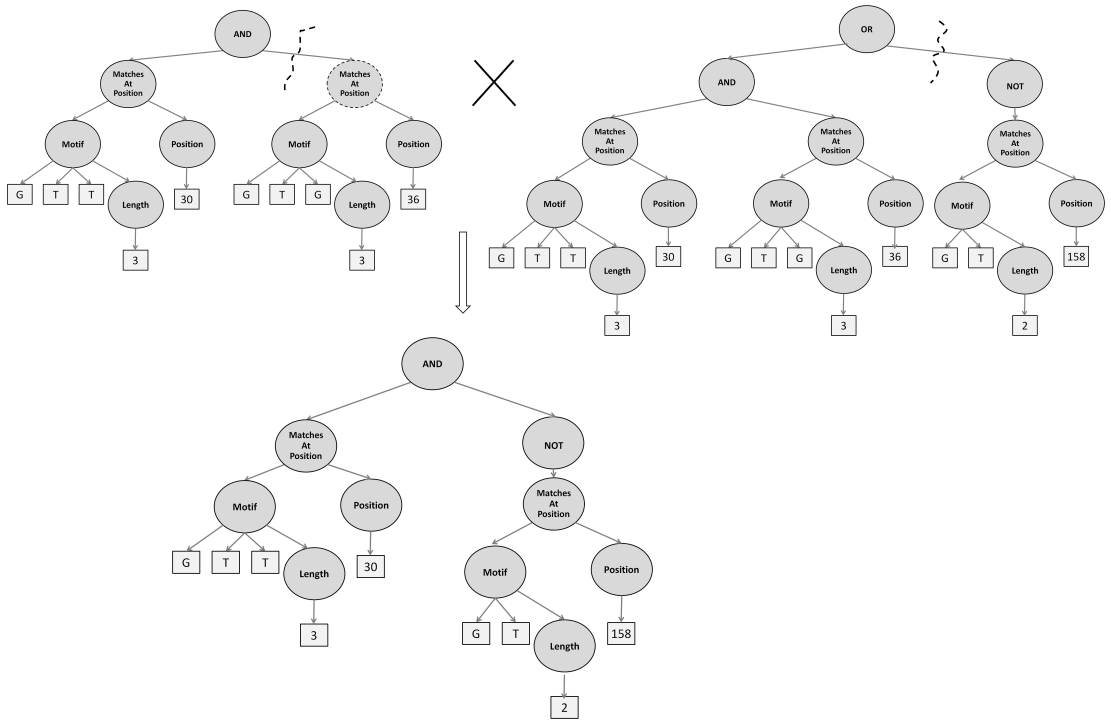

Fig. 3. Crossover: the subtrees at the crossover points are swapped to obtain a new individual.

Bloat Control

A common issue in GP is the unconstrained growth of individuals (trees) without improvements to performance. This growth, referred to as bloat, may be limited by genetic operators that restrict the maximum depth of an individual. In addition, parsimony pressure can be applied to penalize the fitness of larger individuals. FG-EA employs parsimony pressure via lexicographic tournament selection which, given multiple individuals with the same fitness, chooses the individual with the smaller depth [5].

A.1.3 Fitness Function

As noted earlier, to keep the computation time reasonable, FG-EA uses a surrogate fitness function given by: $\text{Fitness}(f) = \frac{C_{+,f}}{C_+} * \text{IG}(f)$ (see paper for a description of each component in this equation). IG is often employed as a criterion of a feature's goodness in machine learning [12]. The IG of a feature f with respect to a class attribute c_i ($c_i \in \{+, -\}$ in binary classification) is the reduction in uncertainty about the value of c_i when f is known. Given m class attributes: $\text{IG}(f) = -\sum_{i=1}^{m} P(c_i) \cdot log(P(c_i)) + P(f) \cdot \sum_{i=1}^{m} p(c_i|f) \cdot$

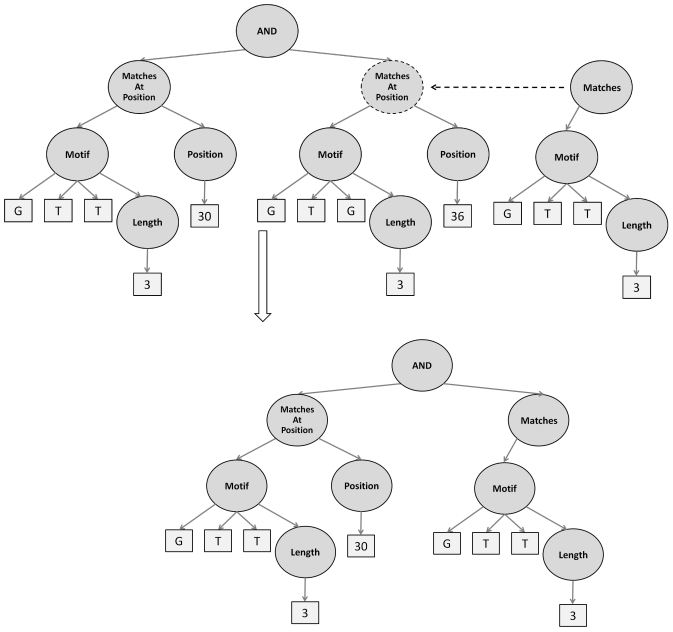

Fig. 2. Mutation: a randomly selected subtree is replaced with a random Grow-generated tree.

$logP(c_i|f) + P(\bar{f}) \cdot \sum_{i=1}^{m} P(c_i|\bar{f}) \cdot log(P(c_i|\bar{f}$. The IG component of the fitness function plays a dual role. First, it eliminates features with IG 0 which do not help the accuracy and precision over the training set. Second, it helps eliminate individuals, such as (`NOT (Matches 'AAAAAA')`), that are formed by negation of other junk individuals. Employing IG in the fitness function is also useful in employing feature reduction at the very end only over the fittest features.

While the goal is to maximize the above fitness function, the Koza fitness for GP aims minimization [5]. Therefore, we define the Koza fitness of a feature f as $\mathrm{Koza}(f) = 1/(\mathrm{Fitness}(f))$. Before selecting the fittest individuals to seed the next generation (as detailed below), FG-EA converts the Koza fitness back into the GP-adjusted fitness $1/(1 + \mathrm{Koza}(f))$. Note that the GP-adjusted fitness takes values in $[0, 1]$.

A.1.4 Hall of Fame and Selecting Elites

The $\ell = 250$ fittest individuals of a generation are added to a hall of fame, which keeps the fittest individuals of each generation. Maintaining a hall of fame guarantees that fit individuals will not be lost or changed through genetic operators. Keeping these individuals in a hall of fame guarantees optimal performance [1]. We employ a hall of fame for two reasons. First, the hall of fame serves as an external memory of the best individuals and allows maintaining diversity in the solution space. Second, the hall of fame represents the solution space at the end of a generational run. Promoting $\ell = 250$ fittest individuals from each generation to the hall of fame results in $6,250$ unique individuals in the hall of fame when FG-EA terminates. A generation seeds its population with $m = 100$ randomly chosen individuals from the current set of features in the hall of fame. This mechanism allows seeding a generation with 100 fit diverse individuals.

A.2 SVM Details

SVMs [10] continue to be popular and successful in a wide variety of binary classification problems. When explicit features are needed, as is the case in this paper, design problems can be summarized in following three steps: (1) map sequence data into a Euclidean vector space; (2) select a kernel function to map the vector space into a higher dimensional and more effective Euclidean space; and (3) tune parameters for the kernel and other SVM parameters (e.g., cost function) to improve performance. The kernel choice is problem-specific and generally determined experimentally.The experiments reported here use a Radial Basis kernel function (RBF), except for worm sequences where LibLinear's fast linear kernel is used. Tuning of the kernel parameters and the SVM cost function is performed through the standard grid search mechanism [9].

A.3 Software Availability

FG-EA was implemented using the standard GP algorithms provided in ECJ [7]. The code was modified to maintain a "hall of fame," a set of the top features generated. In general, default settings were used, as detailed above. RFE was implemented using Weka [11]. Sequence matching and pattern recognition were implemented using BioJava [4]. SVM training and classification was implemented using LibSVM [3] for most experiments. LibLinear [2] was used for the worm data sets. Software and documentation are made available at http://www.cs.gmu.edu/∼ashehu/?q=OurTools.

REFERENCES

[1] C. D. Dosin and R. K. Belew. New methods of competitive coevolution. *Evol. Comput.*, 5(1):1–29, 1997.
[2] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.
[3] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using the second order information for training SVM. *J. Mach. Learn. Res.*, 6(1532-4435):1889–1918, 2005.
[4] R. C. Holland, T. A. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Draeger, A. Yates, M. Heuer, and M. J. Schreiber. BioJava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):2096–2097, 2008.
[5] J. Koza. *On the Programming of Computers by Means of Natural Selection*. MIT Press, Boston, MA, 1992.
[6] S. Luke, G. C. Balan, and L. Panait. Population implosion in genetic programming. in genetic and evolutionary computation. In *Genetic and Evolutionary Computation Conference*, 2003.
[7] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Chircop, J. Compton, W. Haddon, S. Donnelly, B. Jamil, and J. O'Beirne. ECJ: A java-based evolutionary computation research, 2010.
[8] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1993.
[9] C. Staelin. Parameter selection for support vector machines, 2002.
[10] V. N. Vapnik. *Statistical learning theory*. Wiley & Sons, New York, NY, 1998.

[11] Waikato Machine Learning Group. Weka, 2010.
[12] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Intl. Conf. on Mach. Learn.*, pages 412–420. Morgan Kaufmann Publishers, 1997.