Salzburg Interest Group on Integrated Systems

siGis-014-1998 June 30, 1998

Evolutionary Training Data Sets with n–dimensional Encoding for Neural InSAR Classifiers

Special Session on
Evolutionary Computation in Imaging Science
1998 International Conference on Imaging Science, Systems, and Technology
July 6–9, 1998, Las Vegas, Nevada, USA

Helmut A. Mayer
helmut@cosy.sbg.ac.at

Reinhold Huber
reinhold.huber@dlr.de

Department of Computer Science
University of Salzburg, Austria

Aero–Sensing Radarsysteme GmbH
c/o DLR Oberpfaffenhofen, Germany

Correspondence to:

Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Evolutionary Training Data Sets with n–dimensional Encoding for Neural InSAR Classifiers

Helmut A. Mayer
Department of Computer Science
University of Salzburg, Austria
helmut@cosy.sbg.ac.at

Reinhold Huber
Aero–Sensing Radarsysteme GmbH
c/o DLR Oberpfaffenhofen, Germany
reinhold.huber@dlr.de

Abstract *Supervised training of a neural classifier and its performance not only relies on the artificial neural network (ANN) type, architecture and the training method, but also on the size and composition of the training data set (TDS). For the parallel generation of TDSs for a multi–layer perceptron (MLP) classifier we introduce evolutionary resampling and combine (erc) being based on genetic algorithms (GAs). The erc method is compared to various adaptive resample and combine techniques, namely, arc-fs, arc-lh and arc-x4. While arc methods do not consider the classifier's generalization ability, erc seeks to optimize performance by cross-validation on a validation data set (VDS). Combination of classifiers is performed by all arc methods so as to reduce the classifiers' variance, hence, erc also adopts classifier combination schemes. In order to overcome some deficiencies of the traditional approach of mapping bits of GA chromosomes to elements of a set (bit mapping) for evolution of subsets, we investigate the use of n–dimensional encoding. With this approach all available patterns are arranged in an n–dimensional space and the patterns are selected by evolving line segments conveying the data set. All algorithms are compared for a real–world problem, the classification of high resolution interferometric synthetic aperture radar (InSAR) data into several land–cover classes.*

Keywords: InSAR images, neural classifiers, training data set construction, genetic algorithms

1 Introduction

MLPs are regarded as classifiers of low *Bias* and high *Variance*. In order to decrease variance adaptive resampling and combination methods have been suggested [1]. As these schemes are inherently sequential and are not explicitly concerned with the generalizing ability of the classifier, we suggest parallel evolutionary resampling and combine optimizing classifier performance on an independent VDS. A (sub)optimal TDS is evolved in parallel from all available patterns in a *Data Store*. The data store patterns can be preprocessed by feature extraction and selection methods.

Feature extraction and classification of SAR images is commonly based on combined point and texture classification [2]. In this work we used airborne high resolution InSAR data consisting of SAR backscatter image data and InSAR phases correlation (coherence image). Texture extraction is applied to both images prior to classification by a one–hidden–layer MLP. In order to identify feature subsets of most discriminating power, feature selection based on average *Jeffreys–Matusita–Distance* (JMD) has been applied.

2 InSAR Data Preprocessing

A Synthetic Aperture Radar (SAR) is an active microwave instrument providing high–resolution data for remote sensing applications.

Interferometric SAR requires a two–antenna system and provides the ability to derive height information from received signals phase difference and information of the dominant scattering mechanism, which is related to the land–cover, from the measured decorrelation between the signals received by the antennas.

Extraction of textual features from SAR backscatter and InSAR coherence data is based on local statistics of first and second order. Specifically, we employed the coefficient of variation $\hat{C}_v = s/\bar{x}$ (s is the sample standard deviation and $\bar{x}$ the sample mean in a 7×7 window) for texture description by first order statistics. For texture description employing second order statistics, e.g. image covariance structure, we have chosen the method of *Eigenfilters* [3].

The primary features SAR backscatter S and coherence C together with the derived features from local statistics make up a potentially high–dimensional feature vector. The *"Curse of Dimensionality"* [4] and empirical results exploring feature subset selection techniques [5] suggest the construction of a feature vector of lower dimension for a given TDS size. To identify the most relevant features among the large number of texture features derived from *Eigenimages* the average Jeffreys–Matusita–distance (JMD) between classes is evaluated for all possible subsets of an initial subset. Each subset is characterized by its average JMD. Only one feature based on Eigenfilters was selected from SAR and coherence image, respectively, by ranking the subsets by their average JMD [6]. The features of largest frequency in the best subsets were retained, namely, fourth Eigenimage for backscatter E_{S4} and third Eigenimage for coherence E_{C3}.

The result of feature extraction and selection is a 5–dimensional feature vector $\vec{x}$ containing the features SAR backscatter, InSAR coherence, and the above mentioned textural features $\hat{C}_v$, E_{S4} and E_{C3}. Data stores of different size for four different land–cover classes, namely, water, forest, built–up and open area, have been generated randomly from an area of 768×768 pixels of airborne AeS-1 SAR data. The size of the data stores is 200, 2000, and 10000 pixels with an equal number of examples for each class. Additionally, VDSs and test sets containing 2000 examples each with equal class size have been generated by random sampling. Figure 1 shows the five selected feature images.

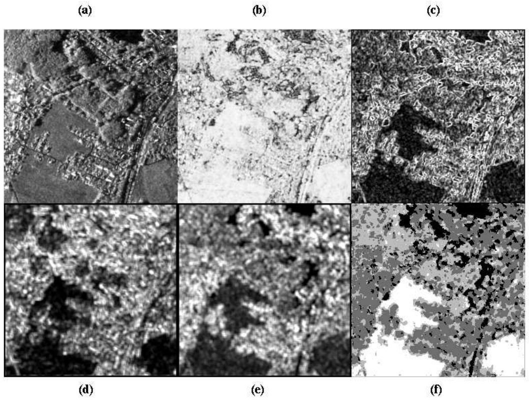

Figure 1: Feature images and multi–layer perceptron classification: (a) SAR image, (b) InSAR coherence, (c) coefficient of variation, texture energy of (d) fourth principal component of SAR image, and (e) third principal component of coherence, (f) classification.

3 Arcing Classifiers

Adaptive resample and combine (ARC) is a technique to increase classifier performance. Various methods and predecessors are described in recent papers, e.g. [7], [1] and [8]. The average error $\mathrm{Error}(C, N, x)$ of a classifier C for sample size N at point x is characterized by a decomposition into bias $\mathrm{Bias}(C, N, x)$ and variance $Var(C, N, x)$ term [9]:

$$\mathrm{Error}(C, N, x) = \mathrm{Bias}(C, N, x)^2 + \mathrm{Var}(C, N, x).$$

Arcing tries to minimize $\mathrm{Var}(C, N, x)$ by voting among classifiers.

All arcing algorithms use a data store $T(\vec{x}, C)$ containing N examples which define an association between a feature vector $\vec{x}$ and M different classes C. The set T serves as the basis for estimating the class conditional densities $P(\vec{x}|C_m)$. Arcing starts by sampling with replacement from T with an initial probability of

$p_1(n) = \frac{1}{N}$. Arcing is iterated for K steps and sampling probabilities are adapted due to specific criteria in order to focus the classifier on "hard to learn" examples. The selection probability of an example n in step k is denoted by $p_k(n)$. We shortly present the operation of the considered arcing methods (all methods start with $k = 1$) in Tables 1, 2, and 3.

Arc-fs (called AdaBoost.M2 in [7])
$\Rightarrow$ Sample with replacement from T with probabilities $p_k(n)$ and construct the classifier C_k using the resampled set T_k. Classify T_k using C_k and let $d(n) = 1$ if example n is incorrectly classified else $d(n) = 0$. Let $e_k = \sum p_k(n)d(n)$ and $\beta_k = (1 - e_k)/e_k$. Update probabilities $p_{k+1}(n) = p_k(n)\beta_k^{d(n)}$. Normalize probabilities $p_{k+1}(n) = p_{k+1}(n)/\sum_i p_{k+1}(i)$. Let $k = k + 1$ and goto $\Rightarrow$ if $k < K$. arc-fs combines $C_1..C_k$ by weighted voting assigning the weight $log(\beta_k)$ to classifier C_k.

Table 1: Pseudocode for arc-fs.

4 Evolutionary Resample and Combine

Based on a method proposed in [10], we employed *Genetic Algorithms* (GA) for the evolutionary search of TDSs optimized towards the classification accuracy of an MLP on a VDS. The patterns constituting a complete TDS are selected in parallel by the GA, the MLP is trained with these individuals (TDSs), and TDS fitness is assigned according to the MLP performance on the VDS. Hence, the quality of single training patterns is not explicitly evaluated, but rather their contribution and cooperation in a TDS for a well generalizing MLP is awarded.

Arc-lh
$\Rightarrow$ Like arc-fs. Let $e_k(n) = \lvert\vec{t}(n) - \vec{o}(n)\rvert^2$ where $\vec{t}(n)$ and $\vec{o}(n)$ are target and output of C_k. Update probabilities $p_{k+1}(n) = p_k(n) + e_k(n)$. Normalize probabilities $p_{k+1}(n) = p_{k+1}(n)/\sum_i p_{k+1}(i)$. Let $k = k + 1$ and goto $\Rightarrow$ if $k < K$. arc-lh combines $C_1..C_k$ by majority voting.

Table 2: Pseudocode for arc-lh.

Depending on the encoding scheme multiple copies of specific training patterns can be selected into the TDS. This is generally valid for the arc approaches focusing on "hard to learn" training patterns. Contrary to arcing, where TDS size is fixed, ercing allows for evolution of TDSs of varying size.

The fitness of a TDS is given by the classification performance of the trained MLP $f = \frac{\sum_N (1-d(n))}{N}$ (overall accuracy, $d(n)$ as with arc-fs). We will use the terms T–fitness and V-Fitness for the classification accuracy on TDSs and VDSs, respectively.

Following the work of [11], where evolved MLPs of the last generation are used for majority voting as opposed to the prevailing practice of utilizing the MLP with the best fitness, we investigated three ways to exploit the knowledge accumulated by evolution. For the classification of a test set *erc–sb* employs the single best MLP discovered, *erc–vb* the g (number of generations) best MLPs, and *erc–vl* all MLPs of the last generation. The MLPs are combined by majority voting.

For TDS evolution experiments presented below we used a generational GA with basic operators implemented in the parallel netGEN system [12] evolving MLP topologies and/or TDSs on an arbitrary number of interconnected workstations.

Arc-x4
$\Rightarrow$ Like arc-fs.
Classify T_k using C_k and let $m_k(n) = \sum_\kappa d_\kappa(n)$ be the number of misclassifications of example n using classifier C_κ for $\kappa \in 1..k$.
Update probabilities $p_{k+1}(n) = 1 + m_k(n)^4$.
Normalize probabilities $p_{k+1}(n) = p_{k+1}(n) / \sum_i p_{k+1}(i)$.
Let $k = k+1$ and goto $\Rightarrow$ if $k < K$.
arc-x4 combines $C_1..C_k$ by majority voting.

Table 3: Pseudocode for arc-x4.

5 TDS Encoding

The traditional set encoding (bit mapping) in GAs is simple and straightforward. Each training pattern in the data store is represented by a single bit of the GA *Chromosome*. Thus, the number of bits (chromosome length l) is equal to the number of training patterns in the data store. The actual TDS is built by selecting all the patterns whose corresponding bits are of value 1. This encoding scheme is able to generate TDSs with a size ranging from $0 - l$ patterns.

Although, for most problems involving a search for optimal subsets, multiple selection of elements might be inconvenient [13], it is by no means clear that a TDS should not contain specific patterns in multiple copies. However, this cannot be achieved by the bit mapping approach. Another drawback of this encoding scheme is the chromosome length increasing linearly with data store size. Moreover, when devising a different scheme by coding indices to the data store, the chromosome length for bit mapping is multiplied by the number of bits used to encode a specific index. While this encoding scheme would allow for multiple copies of patterns in the TDS, the corresponding chromosome length becomes impractical (e.g., a data store size of 10000 results in $l = 140000$).

For these reasons we suggest n–dimensional encoding. (Please, note that the term "dimension" refers to the way the patterns (elements) are arranged in a data structure of a computer program and should not be mingled with the dimension of the vector representing a training pattern.) This scheme can be visualized by putting all data store patterns in a hypercube of dimension n and appropriate volume (e.g., 100 patterns can be arranged in a 10×10 square, a $5 \times 4 \times 5$ box, or a $2 \times 5 \times 2 \times 5$ hypercube). Patterns in these structures are selected for a TDS, if they are positioned on an evolved line segment whose n–dimensional endpoints are encoded on the chromosome. Thus, a single line segment is able to represent a number of patterns. Moreover, the number of bits used to encode an endpoint is reduced when compared to a linear arrangement of patterns. As an example the mechanisms of 2–dimensional encoding are presented in Figure 2.

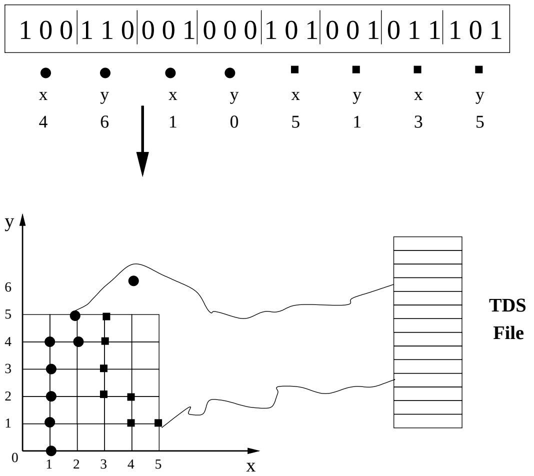

Figure 2: 2–dimensional encoding scheme.

In Figure 2 the patterns are (virtually) located on the intersection of the grid lines of the 5×5 square. The endpoints are encoded using 3 bits, and the "line" segment is generated by a greedy Manhattan walk. All patterns visited (including the endpoints) are put into the TDS File. Points of a segment outside the square (or any n–dimensional hypercube) do not proliferate a TDS pattern.

The n–dimensional encoding scheme enables the evolution of TDS size from $0 - l$ patterns (possibly more than l patterns, but we restricted maximum TDS size to the size of the data store) and allows for multiple copies of patterns. The arrangement of patterns in an n–dimensional structure introduces a spatial dependency. In the following experiments we filled the hypercube (square, box) class by class, i.e., patterns of the same class were clustered. This enhances an efficient sampling of patterns, as specific line segments can evolve in specific classes. Still, if a number of patterns in distant areas would contribute positively to the TDS, separate line segments would have to be evolved for any of these patterns. The chromosome length (number of line segments) has been set corresponding to the minimal number of line segments being necessary to select all data store patterns (e.g., 6 in Figure 2). The side lengths of the hypercube can also be varied (e.g., 6×4 or 3×8), but we were not overly concerned with investigating different combinations of side lengths.

6 Experiments

The following GA and ANN parameters have been used with all the experiments in this paper:

GA Parameters: Population Size = 50, Generations = 50, Crossover Probability $p_c = 0.6$, Mutation Probability $p_m = \frac{1}{l}$, Crossover = 2–Point, Selection Method = Binary Tournament.

ANN Parameters: Network Topology = $5 - 10 - 4$ One–Hidden–Layer Perceptron, Activation Function (all neurons) = Sigmoid, Output Function (all neurons) = Identity, Training = RPROP [14], Learning Parameters $\Delta_0 = 1.0$, $\Delta_{max} = 50.0$, Number of Training Epochs = 1000.

Table 4 and Table 5 summarize the voting performance on the test set by means of overall accuracy and kappa coefficient for arcing and ercing with different encodings, respectively.

Generally, ercing is superior to arcing for data store sizes 200 and 2000. For the data store with 10000 patterns the erc methods perform marginally worse than the investigated arc methods. The n–dimensional encoding scheme performs very well with the small data store (200 patterns) and comparably to bit mapping with the larger stores.

Store	arc–fs		arc–lh		arc–x4	
Size	Acc	κ	Acc	κ	Acc	κ
200	0.328	0.103	0.525	0.367	0.611	0.481
2000	0.681	0.674	0.671	0.561	0.659	0.543
10000	0.724	0.632	0.721	0.627	0.722	0.628

Table 4: Overall accuracy and kappa coefficient κ of the voting performance of arc-fs, arc-lh and arc-x4 on the test set for different data store sizes.

Store	erc–sb		erc–vb		erc–vl	
Size	Acc	κ	Acc	κ	Acc	κ
Bit Mapping						
200	0.638	0.517	0.639	0.519	0.649	0.532
2000	0.716	0.621	0.716	0.621	0.708	0.611
10000	0.715	0.619	0.715	0.619	0.720	0.627
2–Dimensional						
200	0.668	0.557	0.677	0.569	0.678	0.570
2000	0.703	0.603	0.710	0.613	0.707	0.609
10000	0.717	0.623	0.720	0.626	0.722	0.629
3–Dimensional						
200	0.679	0.571	0.688	0.584	0.682	0.576
2000	0.697	0.595	0.716	0.621	0.711	0.615
10000	0.718	0.624	0.719	0.625	0.717	0.622

Table 5: Overall accuracy and kappa coefficient κ of the performance of erc–sb, erc–vb, and erc–vl for different data store sizes using bit mapping, 2–dimensional, and 3–dimensional encoding.

Based on the observation that arcing performance increases remarkably with data store size (Table 4), it can be deduced that the heuristic probability update rules demanding a (re)normalization of probabilities induce a dependency on data store size. This effect is much smaller with ercing (Table 5), although, the search space increases exponentially with data store size.

The arcing performance heavily relies on combining of and voting among classifiers, whereas ercing searches for single good classifiers. Combination of the evolutionary classifiers generates only slightly different results, however, erc-vb never fell below the performance of the single best MLP (erc-sb), while erc-vl occasionally did. Generally, ercing is robust with respect to data store size.

While arcing focuses only on "hard to learn" patterns (arc–lh even more on the difficult patterns generating highest error), ercing concentrates on the best ensemble of patterns. Figures 3, 4, and 5 show the evolutionary development of T– and V–fitness for different encodings.

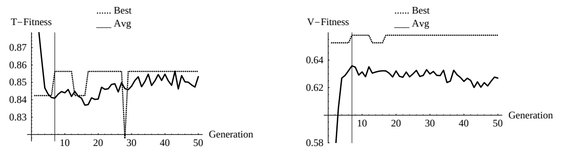

Figure 3: *Bit Mapping* – Best and average T– and V–fitness per generation of TDSs evolved from a data store with 200 patterns.

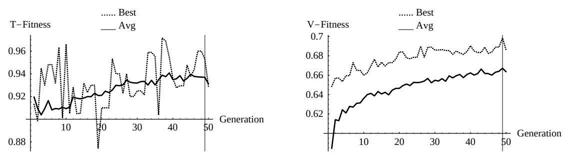

Figure 4: *2–Dimensional Encoding* – Best and average T– and V–fitness per generation of TDSs evolved from a data store with 200 patterns.

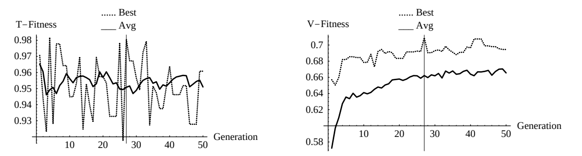

Figure 5: *3–Dimensional Encoding* – Best and average T– and V–fitness per generation of TDSs evolved from a data store with 200 patterns.

Clearly, it can be seen that the average V-fitness increases steadily with n–dimensional encoding (Figures 4 and 5), whereas it decreases with bit mapping (Figure 3), where the best individual is found early on. It seems that as soon the GA focuses on a search region mutation mostly decreases fitness. As the chromosome contains mostly 1s (Table 6), it is more likely that mutation decreases the number of patterns resulting in decreasing average V–fitness, but increasing average T–fitness. With n–dimensional encoding mutations have a more complex effect, as a single bit mutation can change a line segment significantly adding or removing a number of patterns at once. The possible inclusion of multiple copies of patterns in the TDS is suggested by the high T–fitness, as the MLP can be easier trained on a smaller variety of patterns. A detailed statistic of the evolved TDSs is presented in Table 6.

	Size	Id.	h_{c0}	h_{c1}	h_{c2}	h_{c3}
	200	0	0.250	0.250	0.250	0.250
Bit	181	0	0.276	0.276	0.276	0.171
2–D	149	49	0.443	0.201	0.215	0.141
3–D	103	33	0.369	0.311	0.155	0.165
	2000	3	0.250	0.250	0.250	0.250
Bit	1988	3	0.252	0.252	0.252	0.246
2–D	1402	385	0.324	0.288	0.263	0.125
3–D	2000	807	0.259	0.286	0.256	0.200
	10000	2	0.250	0.250	0.250	0.250
Bit	8134	0	0.307	0.307	0.307	0.078
2–D	6297	1682	0.289	0.310	0.265	0.136
3–D	8253	2808	0.259	0.278	0.257	0.207

Table 6: Statistics of evolved TDSs using various encodings and data stores (Id., number of identical patterns, h_{ci}, relative class distribution of patterns).

In comparison to bit mapping the use of n–dimensional encoding results in smaller TDSs with a remarkable amount of identical patterns (the total number of pattern copies). E.g., with data store size 200 a TDS with 103 patterns (33 of them copies) has been evolved using 3–dimensional encoding. Employing this TDS for MLP training resulted in the best observed performance over all arcing and ercing methods. The large variations in the relative class distribution of patterns (Table 6) in the evolved TDSs of similar performance imply that there is no pronounced optimal distribution.

With increasing data store size the performance of bit mapping and n–dimensional encoding is comparable. However, when looking at the graphs for the small data store it might be argued that a higher number of generations will lead to even better solutions, as best and average fitness are still increasing (especially, with 2–dimensional encoding, Figure 4) which is not true for bit mapping.

7 Outlook

Further research with n–dimensional encoding will include studies on the impact of varying hypercube side lengths, arrangement of patterns and their corresponding classes, and different methods to connect the evolved endpoints of a "line" segment (e.g., circle segments or polynomial segments). Also, the use of n–dimensional encoding for problems with an inherent structured arrangement of patterns (e.g., feature extraction from a 2–dimensional image) should be investigated.

8 Acknowledgments

We wish to thank the *Austrian Center for Parallel Computation* (ACPC), Group Salzburg, for making available a cluster of DEC AXP workstations where most of the experiments have been run.

References

[1] Leo Breiman. Arcing the Edge. Technical Report 486, Department of Statistics, University of California, Berkeley, CA, USA, June 1997.

[2] Anne H. Schistad Solberg and Anil K. Jain. Texture Fusion and Feature Selection Applied to SAR Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 35(2):475–479, March 1997.

[3] F. Ade. Characterization of Textures by 'Eigenfilters'. *Signal Processing*, 5:451–457, 1983.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[5] Wojciech Siedlecki and Jack Sklansky. On Automatic Feature Selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197–220, 1988.

[6] Reinhold Huber and Luciano V. Dutra. Feature Selection for ERS 1/2 InSAR Classification: High Dimensionality Case. In *Proceedings of the International Geoscience and Remote Sensing Symposium*, Seattle, WA, USA, July 1998.

[7] Yoav Freund and Robert E. Schapire. Experiments with a New Boosting Algorithm. In *Proceedings of the International Conference on Machine Learning*, 1996.

[8] Friedrich Leisch and Kurt Hornik. ARC-LH: A New Adaptive Resampling Algorithm for Improving ANN Classifiers. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, Cambridge, MA, USA, 1997.

[9] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias / Variance Dilemma. *Neural Computation*, 4(1):1–58, 1992.

[10] Helmut A. Mayer and Roland Schwaiger. Towards the Evolution of Training Data Sets for Artificial Neural Networks. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 663–666. IEEE Press, 1997.

[11] X. Yao and Y. Liu. Ensemble Structure of Evolutionary Artificial Neural Networks. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, pages 659–664. IEEE Press, 1996.

[12] Helmut A. Mayer, Reinhold Huber, and Roland Schwaiger. Lean Artificial Neural Networks - Regularization Helps Evolution. In *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and their Applications*, pages 163–172, 1996.

[13] Nicholas J. Radcliffe and Felicity A. W. George. A Study in Set Recombination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 23–30. University of Illinois at Urbana-Champaign, Morgan Kaufmann, 1993.

[14] Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceedings of International Conference on Neural Networks*, pages 586–591, San Francisco, CA, USA, 1993.