Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition Letters 29 (2008) 796–802
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Classifier ensemble selection using hybrid genetic algorithms

Young-Won Kim [a], Il-Seok Oh [b,*]

[a] Postal Technology Research Center, ETRI, 161 Gajeong-dong, Yuseong-gu, Daejeon, Republic of Korea
[b] Department of Computer Science, Chonbuk National University, Deokjin-dong 664-14, Jeonju, Chonbuk 561-756, Republic of Korea

Received 3 August 2006; received in revised form 10 December 2007
Available online 18 January 2008

Communicated by T.K. Ho

Abstract

This paper proposes a hybrid genetic algorithm for classifier ensemble selection. In this paper, two local search operations used to improve offspring prior to replacement are proposed. The operations are parameterized in order to control the computation time. Experimental results and statistical tests demonstrate the effectiveness of the proposed hybrid genetic algorithm and related local search operations.

Keywords: Multiple classifier combination; Ensemble selection; Genetic algorithm; Local search operation

1. Introduction

Research relating to combining multiple classifiers is abundant. The success in improving recognition performance has been revealed by theoretical and/or empirical studies (Kittler et al., 1998; Ho, 2002). In assuming availability of the original set of multiple classifiers (this set is referred to as a classifier pool), Ho raised two major issues, referred to as *coverage optimization* and *decision optimization* (Ho, 2002).

–*Coverage optimization:* The problem of selecting an optimal classifier subset (called classifier ensemble) from a given classifier pool.
–*Decision optimization:* The problem of combining outcomes of the classifiers belonging to a given classifier ensemble.

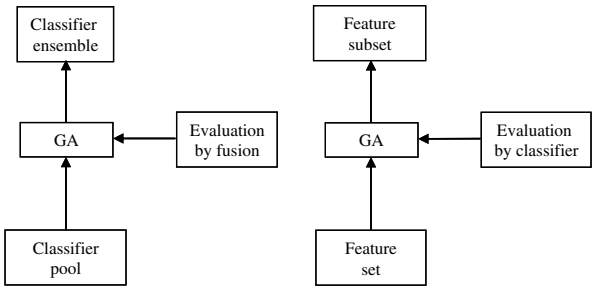
Researchers have placed much more effort on the decision optimization issue (Ueda, 2000; Fumera and Roli, 2005). However, coverage optimization is not of lesser importance than decision optimization, because outcomes of the classifier ensemble are directly input into the combination algorithm (Roli and Giacinto, 2002). Classifier ensemble selection is defined to be the problem of selecting a classifier ensemble of d classifiers, from a classifier pool of N classifiers. The problem has a large search space with a size of ${}_N\mathrm{C}_d$.

Our brief review on recent researches revealed that the coverage optimizations are getting more and more attention. There are many approaches, such as bagging and boosting using the partitioned training subsets (Theodoridis and Koutroumbas, 2006), methods using the different feature subsets (Rodriguez et al., 2006), adaptive methods considering simultaneously the classifier ensemble construction and classifier training (Nicolas et al., 2005; Wanas et al., 2006). Comparative studies are also available (Banfield et al., 2007; Sohn and Shin, 2007). In order to obtain excellent quality solutions in a reasonable computation time, genetic algorithms have recently been used (Zhou et al., 2002; Oliveira et al., 2003; Wang and Wang, 2006). In spite of their success, room remains to improve solution quality.

The aim of this paper is to improve the searching capability of a genetic algorithm, in the context of classifier ensemble selection. The strategy is to develop a hybrid

* Corresponding author. Tel.: +82 63 2703401; fax: +82 63 2702394.
E-mail address: isoh@chonbuk.ac.kr (I.-S. Oh).

0167-8655/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2007.12.013



(a) Classifier ensemble selection (b) Feature selection

Fig. 1. Analogy between ensemble selection and feature selection.

```
Hybrid_GA() {
    initialize population P;
    repeat {
        for i=1 to k {
            select two parents p1 and p2 from P;
            offspring_i = crossover(p1, p2);
            offspring_i = mutation(offspring_i);
            local_search(offspring_i);
        }
        replace offspring_1, ..., and offspring_k in P;
    } until (stopping_condition);
    return the best chromosome;
}
```

Fig. 2. Hybrid genetic algorithm.

genetic algorithm, by embedding local search operations in the simple genetic algorithm. Two local search operations are proposed; sequential and combinational.

Fig. 1 shows the analogy between the ensemble selection problem and feature selection problem. The framework of problem solving is the same. So this paper adopts the hybrid genetic algorithm (HGA) in (Oh et al., 2004) proposed for the feature selection. In this paper, we propose newly the combinational search operation in order to improve further the search capability of the HGA.

Objective experimental setup was designed using several well-known datasets, including a large spectrum of search space sizes. Experimental results demonstrated that the proposed hybrid genetic algorithm had superior searching capability over the simple genetic algorithm, and the combinational search operation produces a superior solution, compared to the sequential search operation.

2. Hybrid genetic algorithms (HGA)

Fig. 2 presents an outline of the proposed genetic algorithm. The parameter k is the generation gap, where $k = 1$ represents the steady-state procedure and $k = |P|$ represents the generational procedure. $|P|$ represents the population size. A notable difference from the simple genetic algorithm is the embedding of a local search operation, after the genetic operations (crossover and mutation). Removal of the function calling, *local_search(offspring$_i$)* results in the simple genetic algorithm (SGA). Detailed specification of the algorithms and parameter settings are presented in Section 2.1. Due to their importance in the proposed HGA, the local search operations are separately described in Section 2.2.

2.1. Algorithm description

(1) *Chromosome encoding*: Each of the N classifiers in the classifier pool has a unique number from 1 to N. A string with N binary digits is used, where the ith classifier is assigned to the ith digit. A digit has one of two values, 1 and 0, meaning *selected* and *removed*, respectively. For example, chromosome C = 1000100101 (when $N = 10$) means that the first, fifth, eighth and tenth classifiers are selected. Two operations are defined, X(C) and Y(C), which transforms the binary string into sets. The X(C) and Y(C) operations are the selected and removed classifiers, respectively. In the example, $X(C) = \{1, 5, 8, 10\}$ and $Y(C) = \{2, 3, 4, 6, 7, 9\}$. X(C) is the classifier ensemble.

(2) *Initial population*: The initial population is generated by random number generation, in such a way that the desired number of selected classifiers in a chromosome is d.

(3) *Selection*: A chromosome C is evaluated using the recognition rate produced by combining outcomes of the classifier ensemble X(C). Majority voting for the classifier ensemble combination is used. The recognition rate of the combination is denoted by V(X(C)). The selection algorithm uses a rank-based roulette-wheel scheme. The chromosomes in the population are sorted in nonincreasing order, in terms of the associated function values, V(X(C)), and the ith chromosome is assigned fitness using a linear function, $f(i) = max + (i - 1)*(min - max)/(|X(C)|\text{-}1)$ where the *min* and *max* are user-specified parameters. A large gap between *min* and *max* enforces stronger selection pressure, i.e., giving more preference to the better chromosomes.

(4) *Crossover and mutation*: The standard crossover and mutation operators are used. An m-point crossover operator is used, which chooses m cutting points at random and alternately copies each segment out of the two parents. Between two offspring, one is randomly chosen, and the mutation is applied to the chosen offspring. It is important to note that crossover and mutation are likely to produce an offspring whose $|X(C)|$ is not d. In the local search operation, the offspring has the opportunity to be repaired.

(5) *Replacement*: If the resulting offspring is superior to both parents in terms of the fitness, it replaces the similar parent; if it is in between two parents, it

replaces the inferior parent; otherwise, the most inferior chromosome in the population is replaced.

2.2. Local search operations

Genetic algorithms are able to escape from local optima by means of the crossover and mutation operators, and explore a wide range of search space when the selection pressure is properly controlled. However, the algorithms are weak with regard to fine-tuning near local optimum points, resulting in a long running time. In order to improve the fine-tuning capability of SGA, HGAs have been developed in many applications, including feature selection (Oh et al., 2004), the traveling salesman problem (Jog et al., 1989), graph partitioning (Bui and Moon, 1996), and image compression (Zheng et al., 1997). It is probable that the offspring generated after the crossover and mutation operations inherit the parents' good characteristics and are either superior or inferior to their parents. In HGA, prior to the replacement stage, the offspring are given the chance to be improved using local search operations.

A major issue in realizing HGA is to choose the proper operations for local improvement. Two local search operations are developed; sequential and combinational. The two objectives of the operations are to possibly improve the offspring, and repair the offspring to have d 1-bits.

(1) Sequential search operations (SSO)
The operation is outlined below.

```
sequential_local_search(C) {
   n = |X(C)|; // number of 1-bits
   switch {
      case n = d: ripple_rem(r); ripple_add(r);
      case n < d: repeat ripple_add (r) d-n times;
      case n > d: repeat ripple_rem (r) n-d times;
   }
}
```

The primitive operations *rem* and *add*, and composite operations are defined below. The operations, *ripple_rem* and *ripple_add* were first developed for the feature selection problem in (Oh et al., 2004).

rem: Choose the least significant classifier x in X(C), such that $x = \text{argmax}_{i \in X(C)} V(X(C)-\{i\})$, and change the bit of x from 1 to 0.
add: Choose the most significant classifier y in Y(C) such that $y = \text{argmax}_{i \in Y(C)} V(X(C) \cup \{i\})$, and change the bit of y from 0 to 1.
REM(k): Repeat operation *rem*, k consecutive times.
ADD(k): Repeat operation *add*, k consecutive times.
ripple_rem(r) $\equiv \{REM(r);\ ADD(r-1);\}$, $r \geqslant 1$.
ripple_add(r) $\equiv \{ADD(r);\ REM(r-1);\}$, $r \geqslant 1$.

The operations operate with a current classifier ensemble, X(C), represented by a chromosome, and perform local searches around the current solution, by removing the least significant features or adding the most significant features. The searching for the *rem* (*add*) is in the steepest descent (ascent) direction.

The *ripple_rem*(r) procedure first removes the least significant features r times and then adds the most significant features $r-1$ times, resulting in the removal of one feature. Likewise, *ripple_add*(r) increases the size of X by 1. The parameter r is called the *ripple factor*. The ripple factor is used to control the strength of local improvement. The larger the ripple factor, r, the stronger the local improvement obtained.

(2) Combinational search operations (CSO)
In sequential operations, only one classifier is considered at a time, in deciding the most or least significant classifier. Therefore, mutual effects among multiple classifiers are ignored. In the combination search operation, multiple classifiers are considered concurrently. The operation follows.

```
combinational_local_search(C) {
   n = |X(C)|;
   switch {
      case n = d: cadd(s); crem(s);// s is a small number
      case n < d: cadd(d-n);
      case n > d: crem(n-d);
   }
}
```

crem(k): Generate all the subsets with size k from the set X(C), and choose m subsets randomly. Among the chosen m subsets, choose the least significant subset and change the genes in the chosen subset from 1 to 0.
cadd(k): Generate all the subsets with size k from the set Y(C), and choose m subsets randomly. Among the chosen m subsets, choose the most significant subset and change the genes in the chosen subset from 0 to 1.

Table 1
Experimental datasets and performance evaluation

Datasets	Train set size	Test set size	Validation	Comparing SGA and HGA
Letters[a]	15000	5000	Leave-one-out	Performance on test set
Ionosphere	351	0	Leave-one-out	Validation performance
Sonar	208	0	Leave-one-out	Validation performance
WDBC	569	0	Leave-one-out	Validation performance
CENPARMI	4000	2000	Leave-one-out	Performance on test set

[a] The original set of 20000 samples partitioned into 15000 train and 5000 test samples.

Table 2
Performance of 1-NN classifiers in the classifier pool

Datasets	Original feature set size	Classifier pool			
		Feature subset size	Minimum recognition rate (%)	Maximum recognition rate (%)	Average recognition rate (%)
Letter	16	8	85.01	92.29	87.36
WDBC	30	15	90.16	93.67	91.51
Ionosphere	34	17	84.33	90.59	87.94
Sonar	60	30	85.09	87.50	86.18
CENPARMI	512	256	92.90	95.10	94.02

In the operations, only a part of all the subsets is searched, in order to have control over the computation time. A search ratio q, such that $q = m/|_{\mathrm{X}}|\mathrm{C}_{\mathrm{k}}$, can be defined, where $|_{\mathrm{X}}|\mathrm{C}_{\mathrm{k}}$ is total number of subsets.

3. Experiments

In order to construct a classifier pool, 50 1-NN (nearest neighbor) classifiers were made using different feature subsets chosen randomly. We chose 1-NN as the base classifier since it is stable, that is, it is not sensitive to the initial setting. For example, neural networks may not be proper for performance comparison since they produce different recognition rates depending on the initial weight settings. For the combination method, we chose the majority voting since it is the simplest scheme.

Since the experiments aim at comparing the performance between SGA and HGA, the experimental setting should be carefully designed. We chose five datasets shown in Table 1, four from UCI repository (Murphy and Aha, 1994) and one consisting of CENPARMI handwritten numerals. They have been widely used in the pattern recognition community. The table also shows how we compare the SGA and HGA by presenting the validation and test methods.

Table 2 presents various classifier statistics. Each feature subset is half the size of the original feature set. To be fair between SGA and HGA, all the conditions are set to be the same in whole stage of the experimentation, including the running time.

The following parameter setting was used for the datasets in the experiments. For SSO, we adopted the value $r = 2$ recommended in (Oh et al., 2004). It is a difficult job to identify an optimal setting of the involving parame-

Table 3
Comparisons of recognition performance

Datasets	Classifier pool size (N)	Classifier ensemble size (d)	Average (maximum) in %			Statistical significance	
			SGA (a)	HGA with SSO (b)	HGA with CSO (c)	p-Value	Multiple comparisons
Letter	50	4	95.89 (95.93)	96.30 (96.33)	96.65 (96.66)	0.00**	$a < b < c$
		8	96.87 (96.95)	97.40 (97.42)	97.66 (97.68)	0.00**	$a < b < c$
		12	97.16 (97.19)	97.67 (97.68)	97.83 (97.84)	0.00**	$a < b < c$
		16	97.28 (97.32)	97.80 (97.81)	97.94 (98.00)	0.00**	$a < b < c$
		20	97.32 (97.34)	97.83 (97.85)	97.92 (97.94)	0.00**	$a < b < c$
		24	97.35 (97.37)	97.84 (97.86)	97.98 (97.98)	0.00**	$a < b < c$
WDBC	50	4	95.11 (95.43)	95.36 (95.43)	95.15 (95.43)	0.08	–
		8	95.61 (95.78)	95.92 (95.96)	95.99 (96.14)	0.00**	$a < b, c$
		12	95.47 (95.61)	95.89 (95.96)	95.96 (95.96)	0.00**	$a < b, c$
		16	95.43 (95.61)	95.68 (95.78)	95.89 (95.96)	0.01*	$a < b < c$
		20	95.39 (95.61)	95.61 (95.78)	95.64 (95.78)	0.00**	$a < b, c$
		24	95.33 (95.43)	95.50 (95.61)	95.64 (95.78)	0.00**	$a < b < c$
Ionosphere	50	4	93.16 (93.45)	93.45 (93.45)	93.45 (93.45)	0.03*	$a < b, c$
		8	93.96 (94.02)	94.53 (94.59)	94.59 (94.59)	0.00**	$a < b, c$
		12	94.36 (94.59)	94.70 (94.87)	95.16 (95.16)	0.00**	$a < b < c$
		16	94.36 (94.59)	94.76 (94.87)	94.93 (95.16)	0.00**	$a < b, c$
		20	94.25 (94.59)	94.70 (94.87)	94.87 (94.87)	0.01*	$a < b, c$
		24	94.19 (94.59)	94.53 (94.87)	94.81 (94.87)	0.00**	$a < b, c$
Sonar	50	4	91.06 (91.83)	91.83 (91.83)	91.83 (91.83)	0.00**	$a < b, c$
		8	91.35 (91.83)	92.34 (92.79)	92.59 (92.79)	0.00**	$a < b, c$
		12	91.73 (92.31)	92.30 (92.79)	92.69 (92.79)	0.00**	$a < b, c$
		16	91.15 (91.83)	92.40 (92.79)	92.69 (92.79)	0.00**	$a < b, c$
		20	90.96 (91.35)	91.83 (92.31)	92.31 (92.31)	0.00**	$a < b < c$
		24	90.67 (90.87)	91.54 (91.83)	91.83 (91.83)	0.00**	$a < b, c$
CENPARMI	50	4	96.69 (96.70)	96.90 (96.90)	97.00 (97.00)	0.00**	$a < b < c$
		8	97.02 (97.10)	97.22 (97.30)	97.40 (97.40)	0.00**	$a < b < c$
		12	97.06 (97.10)	97.30 (97.30)	97.40 (97.40)	0.00**	$a < b < c$
		16	97.06 (97.10)	97.26 (97.30)	97.38 (97.40)	0.00**	$a < b < c$
		20	97.06 (97.10)	97.22 (97.30)	97.36 (97.40)	0.00**	$a < b < c$
		24	96.98 (97.10)	97.18 (97.20)	97.30 (97.30)	0.00**	$a < b < c$

* $p < 0.05$.
** $p < 0.01$.

ters (Eiben et al., 1999). Our experiment has not attempted a systematic optimization process. Tuning their values may lead to a performance improvement.

Parameter setting

$k = 1$(steady-state procedure)
$|P|$ (population size) $= 20$
min and *max* (in rank-based selection) $= 0.1$ and 1.0
m (in m-point crossover) $= 3$
p_m (mutation rate) $= 0.1$
r (in SSO) $= 2$
q (in CSO) $= 0.5$
s (in CSO) $= 3$

Table 3 and Fig. 3 present the performance comparison between SGA and HGA, and between SSO and CSO. For the classifier pool with $N = 50$, Table 3 shows the ensembles with size 4, 8, 12, 16, 20 and 24. The Fig. 3 shows visually the ensembles for all sizes. The programs are run five times independently. Their average and maximum rates are presented.

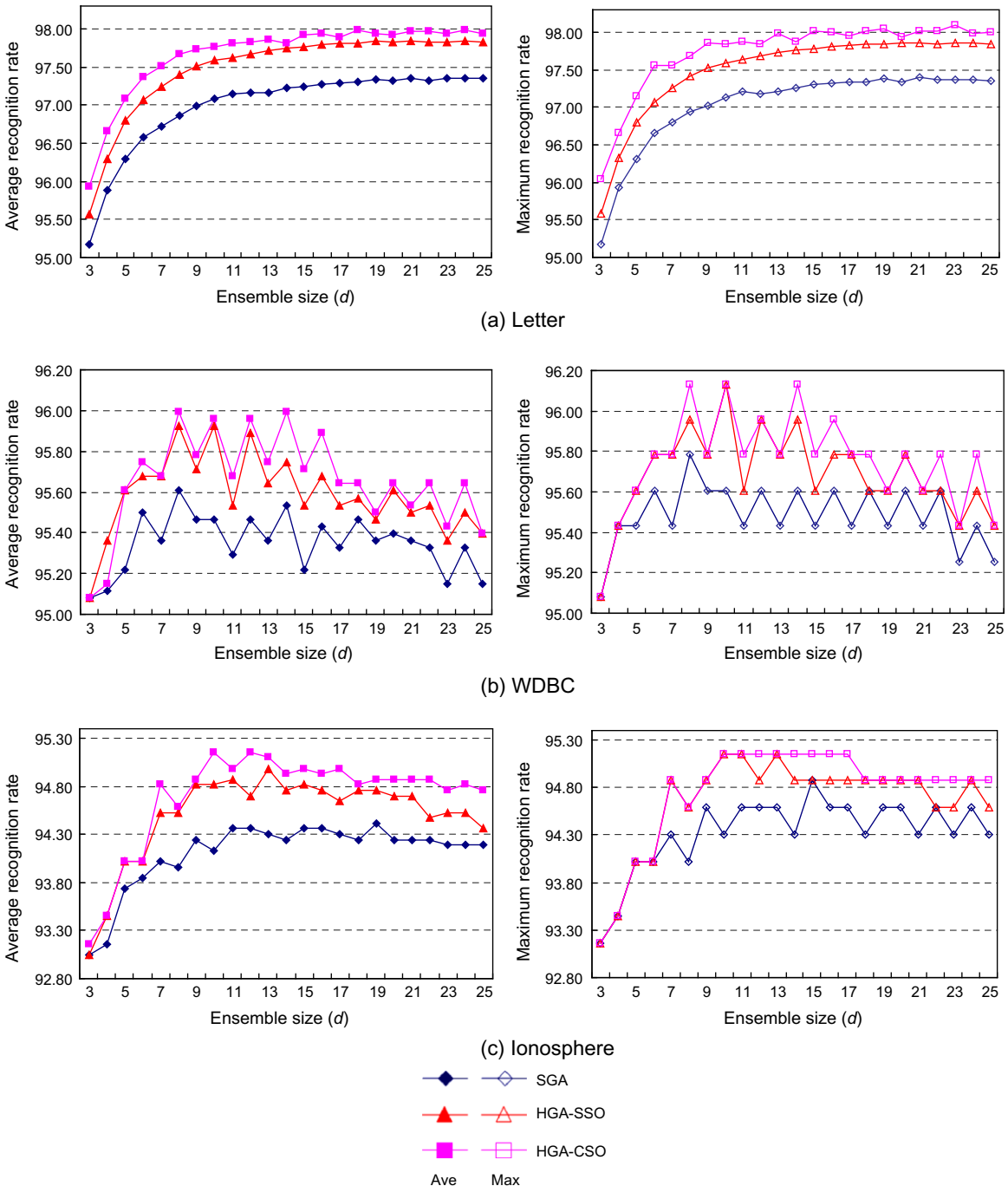


Fig. 3. Comparisons of recognition performance.

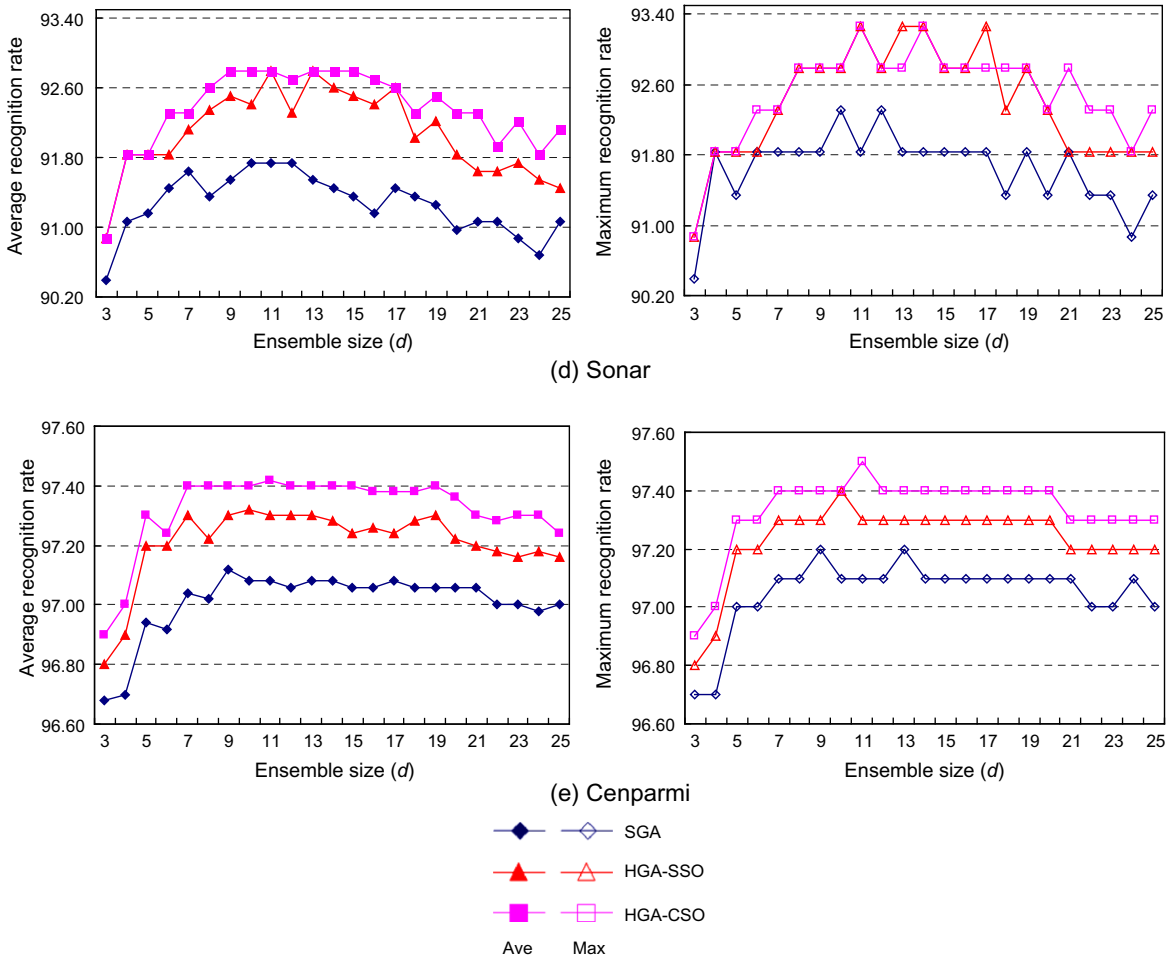


Fig. 3 (*continued*)

We observe that the classifier ensembles in Table 3 had better recognition rates than the single best classifier in Table 2. For example, in Sonar, the GAs found a solution of $d = 4$ with recognition rate 91.83% while the single best classifier had 87.50%. This explains that the ensemble method is effective even if we use the simple voting combination scheme.

We will compare three algorithms with the averages and maximum and show that their differences are statistically significant. The averages and maximum in Table 3 and Fig. 3 demonstrate that HGA produced better results than SGA consistently over various values of d. The exceptions found in Table 3 were the cases for the maximum rates in WDBC, ionosphere, and Sonar when d is small, i.e., $d = 4$. The gap is 0.25–1.54 and 0.00–0.96 for the average and maximum manners, respectively. The comparison of SSO and CSO used by HGA revealed that CSO outperformed SSO consistently with a few exceptions. The exceptions were found at $d = 4$ for WDBC, Ionosphere, and Sonar. The exceptions above seem to occur for the problems with small search space which any algorithm can easily solve.

We also measured the statistical significance probability for data from the five independent runs. The last two columns present the results. The ANOVA analysis for the test of the difference of the three methods was done using SPSS 12.0. All the cases except WDBC for $d = 4$ showed the statistically significant differences. For the significant cases, we further performed the multiple comparisons (post hoc) using LSD technique. The results are shown in the last column. Based on all of these data and analysis, we recommend the HGA with CSO as the most promising algorithm.

Another observation can be made relating to the size of the best classifier ensemble. The graphs in Fig. 3 show that the appropriate size of the classifier ensemble in Letter was around 20. For other datasets, it was around 10. This demonstrates that a small classifier ensemble may suffice in obtaining an optimal combination.

Our last comparison was done for the stability of the algorithms. For this purpose, we measured the standard deviation of the five independent runs for each of algorithms. Table 4 shows the results. The smaller value means the more stable behavior. Our conclusion is that HGA is more stable than SGA, and CSO is more stable than SSO. The CSO beat SSO more frequently.

4. Conclusions

For classifier ensemble selection, hybrid genetic algorithms along with two local search operations, are proposed.

Table 4
Standard deviation × 1000

Datasets	Classifier pool size (N)	Classifier ensemble size (d)	SGA	HGA	
				SSO	CSO
Letter	50	4	0.35	0.34	0.11
		8	0.48	0.26	0.22
		12	0.34	0.26	0.11
		16	0.30	0.10	0.33
		20	0.20	0.23	0.09
		24	0.19	0.12	0.05
WDBC	50	4	1.93	1.57	1.57
		8	2.15	0.79	0.79
		12	1.93	0.96	0
		16	1.24	0.96	0.96
		20	1.47	1.24	0.79
		24	0.96	0.96	0.79
Ionosphere	50	4	2.85	0	0
		8	1.27	1.27	0
		12	1.27	1.56	0
		16	1.27	1.56	1.27
		20	2.38	1.56	0
		24	2.55	3.12	1.27
Sonar	50	4	4.30	0	0
		8	5.89	2.83	2.64
		12	4.02	3.40	2.15
		16	4.30	2.15	2.15
		20	4.02	3.40	0
		24	2.63	2.63	0
CENPARMI	50	4	0	0	0
		8	0.45	0.45	0
		12	0.55	0	0
		16	0.55	0.55	0.45
		20	0.55	0.45	0.55
		24	0.84	0.45	0

Through experiments, it was proven that HGA demonstrated superior searching capability over SGA, and combination search operations are superior over sequential search operations. In the future, several issues will be studied, including the construction of a high-quality classifier pool both in diversity and individual goodness, and the optimal size parameter identification of classifier pool and classifier ensemble.

References

Banfield, R.E., Hall, L.O., Bowyer, K.W., Kegelmeyer, W.P., 2007. A comparison of decision tree ensemble creation methods. IEEE Trans. Pattern Anal. Machine Intell. 29 (1), 173–180.

Bui, T.N., Moon, B.R., 1996. Genetic algorithm and graph partitioning. IEEE Trans. Comput. 45 (7), 841–855.

Eiben, A.E., Hinterding, R., Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. IEEE Trans. Evolut. Comput. 3 (2), 124–141.

Fumera, G., Roli, F., 2005. A theoretical and experimental analysis of linear combiners for multiple classifier systems. IEEE Trans. Pattern Anal. Machine Intell. 27 (6), 942–956.

Ho, T.K., 2002. Multiple classifier combination: lessons and next steps. In: Bunke, H., Kandel, A. (Eds.), Hybrid Methods in Pattern Recognition. World Scientific, pp. 171–198.

Jog, P., Suh, J., Gucht, D., 1989. The effect of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In: Proc. Internat. Conf. on Genetic Algorithms, pp. 110–115.

Kittler, J., Hatef, M., Duin, R.P.W., Matas, J., 1998. On combining classifiers. IEEE Trans. Pattern Anal. Machine Intell. 20 (3), 226–239.

Murphy, P.M., Aha, D.W., 1994 UCI repository for machine learning databases. Technical Report, Department of Information and Computer Science, University of California, Irvine, 1994, <http://www.ics.uci.edu/~mlearn/databases/>.

Nicolas, G.P., Cesar, H.M., Domingo, O.B., 2005. Cooperative coevolution of artificial neural network ensembles for pattern classification. IEEE Trans. Evolut. Comput. 9 (3), 271–302.

Oh, I.S., Lee, J.S., Moon, B.R., 2004. Hybrid genetic algorithms for feature selection. Trans. Pattern Anal. Machine Intell. 26 (11), 1424–1437.

Oliveira, L.S., Sabourin, R., Bortolozzi, F., Suen, C.Y., 2003. Feature selection for ensembles: a hierarchical multi-objective genetic algorithm approach. In: Proc. Internat. Conf. on Document Analysis and Recognition, Edinburgh, pp. 676–680.

Rodriguez, J.J., Kuncheva, L.I., Alonso, C.J., 2006. Rotation forest: a new classifier ensemble method. IEEE Trans. Pattern Anal. Machine Intell. 28 (10), 1619–1630.

Roli, F., Giacinto, G., 2002. Design of multiple classifier systems. In: Bunke, H., Kandel, A. (Eds.), Hybrid Methods in Pattern Recognition. World Scientific, pp. 199–226.

Sohn, S.Y., Shin, H.W., 2007. Experimental study for the comparison of classifier combination methods. Pattern Recognition 40, 33–40.

Theodoridis, S., Koutroumbas, K., 2006. Pattern Recognition, third ed. Academic Press.

Ueda, N., 2000. Optimal linear combination of neural networks for improving classification performance. IEEE Trans. Pattern Anal. Machine Intell. 22 (2), 207–215.

Wanas, N.M., Dara, R.A., Kamel, M.S., 2006. Adaptive fusion and cooperative training for classifier ensembles. Pattern Recognition 39, 1781–1794.

Wang, X., Wang, H., 2006. Classification by evolutionary ensembles. Pattern Recognition 39 (4), 595–607.

Zheng, X., Julstrom, B.A., Cheng, W., 1997. Design of vector quantization codebooks using a genetic algorithm. In: Proc. IEEE Internat. Conf. on Evolutionary Computation, pp. 525–529.

Zhou, Z.H., Wu, J., Tang, W., 2002. Ensembling neural networks: many could be better than all. Artif. Intell. 137, 239–263.