Scout Algorithms and Genetic Algorithms: A Comparative Study

Fabio Abbattista[1], Valeria Carofiglio[1], Mario Köppen[2]

[1] Dipartimento di Informatica
Universita' di Bari
Via E. Orabona 4, 70126 Bari, Italy
email: fabio@di.uniba.it
[2] Department Pattern Recognition
Fraunhofer IPK-Berlin
Pascalstr. 8–9, 10587 Berlin, Germany
email: mario.koeppen@ipk.fhg.de

Abstract. This paper gives a comparative study of the recently proposed Scout algorithm and the standard genetic algorithm (SGA). Although these evolutionary algorithms are differently configurated and use different evolutionary operators, essential similarities of both algorithms can be pointed out. These similarities can be found at three levels, theoretically and empirically: it is discussed that the Scout algorithm fulfills Holland's paradigm of admissible detector configuration; from the definition of the Scout algorithm and its variants, the Schemata theorem for Scout algorithms is justified; and experimental evidence is given, that Scout algorithm's performance reflects SGA's performance for SGA hard problems (Tanese functions) and SGA deceptive problems (Royal Road Functions). Thus, Scout algorithms and SGA are both unique instances of a broader, yet unknown class of evolutionary algorithms. The most important advantage of Scout algorithms is the fashioning of their update rules. This update rule has no counterpart in the SGA.

1 Introduction

In a study on goal-directed pattern recognition [4], Holland discussed the approach to feature detection of Samuel's famous checkerplayer program. Then, Holland derived the basic concepts for a new class of search algorithms, the class which became later on popular as "genetic algorithms" [5]. The basic idea of the study can be summarized in the *admissible detector configuration paradigm*: whenever a goal attaining procedure makes use of a set of feature detectors, this procedure should be able to reconfigure the set of feature detectors as well. From this, two objectives of such procedures can be fulfilled: "*relevance of the detectors and consistency of the model and the value-estimator [nowadays called fitness function – M.K.].*" ([4], p. 292). Also, the goal attaining procedure can be considered as a sampling plan in a "*new and distinct sample space, the space of all admissible detectors...*" (idem., p. 294).

Assuming a procedure, which has access to the set of *all* admissible feature detectors at once, and denoting the actual use of one of these feature detectors

by a binary value, the bitstring notation was born, which later gave the representation of an individual of the evolving population of the standard genetic algorithm (SGA).

The basic problem for finding such a sampling plan was formulated as the facilitation of a measure for the apportionation of credit for errors. Holland qualifies measures of "*average excesses*" as useful. Later, implicit parallelism and the schemata theorem will give the theoretical foundation of those measures. But note that usually those measures are not *directly* used in a genetic algorithm.

What seldom had been considered in subsequent works was the question, if there exist other algorithms, which fulfill the admissible detector configuration paradigm as well. Since all properties of genetic algorithms can be derived from this paradigm, other algorithms based on it should resemble genetic algorithms, too.

One of the most important aspects of genetic algorithms is their versatility. By modifying the selection, crossover or mutation scheme of the standard genetic algorithm, or by maintaining other generation control schemes (as e.g. for the steady-state approach or the inclusion of ancestry), completely new algorithms can be derived. These algorithms can be applied in certain circumstances, when standard genetic algorithms perform poorly.

However, all of these variations base directly on the standard genetic algorithm. Other procedures, different from genetic algorithms, but following the detector configuration paradigm, are expected to provide new insights into the theoretical background of evolutionary algorithms, and to allow the crafting of new, versatile algorithms as well.

This paper presents such an algorithm, the recently proposed Scout algorithm [1]. The Scout algorithm maintains a set of so-called probability vectors. The name "Scout" refers to an exploring entity, which reminds one of a scout that pioneers a pathway in an unknown territory. In the language of the admissible detector paradigm: a Scout, which is represented as a vector of probability values (instead of binary values, as for genetic algorithms), configures a subset of the feature detectors (a bitstring) by means of the probabilities for each single detector. The higher the component i of the probability vector of a Scout, the higher the probability of setting the corresponding bitvalue to 1 (means: using detector i in the plan for that algorithm step). The Scout algorithm apportionates credit for success to the probabilities by the so-called *update rule*, thus employing a *direct* measure of average excess.

The Scout algorithm itself operates on a set of such Scouts. Using a multitude of Scouts does not directly improve the "implicit parallelism" of the sampling plan, but makes the value-estimators more robust.

From this general approach of the Scout algorithm, its fulfillment of the configurable detector paradigm is obvious. It differs from the genetic algorithm in just two points: instead of specifying the chosen detectors directly (by bitstrings), it maintains a "pre-instance," from which detector-specifying bitstrings are instantiated whenever they are needed; and the "average excess" of a Scout's bitstring fitness is directly used for modifying its probabilities.

Section 2 recalls the Scout algorithm and gives two new update rules, which were designed in order to reveal the similarity to genetic algorithms. Then, in section 3, the Schemata theorem for Scout algorithms is justified. Section 4 then gives an experimental validation of the same fact, using test problems, which have been intensively studied for standard genetic algorithms in the past: the Royal Road Functions as an example for a deceptive problem, and the Tanese functions, for which the poor performance of genetic algorithms has puzzled the scientific community some years ago. The paper is followed by a discussion of the results and concludes with the reference.

2 The Scout Algorithm

The Scout algorithm, in its original implementation [1], explores binary search spaces, represented by all the possible binary strings $S = (S_i)$, $i = 1, 2, \ldots, l$ (l is a feature of the problem). The basic element of the algorithm is represented by the probability vector $P = (P_i)$, in which each element P_i gives the probability that the i-th element S_i will be 1. The main goal of the Scout is to identify a vector $P^{opt} = (P_i^{opt})$ in such a way that, applying the P_i^{opt} it is possible to generate an optimal solution, $S^{opt} = (S_i^{opt})$, for the problem at hand. The vector P^{opt} is produced by iterating the Scout algorithm for a prefixed number of cycles, denoted by `Max_Cycles`. For each cycle c, the vector P is left unchanged (it will be denoted with $P(c)$); at the beginning of a run, the values of $P_i(0)$ are all set equal to 0.5. The algorithm is iterative and, in each cycle, it performs the following operations:

1. **Generation**. Generation of K solutions S using $P(c)$. The value of K is constant in all cycles.
2. **Selection**. Evaluation of each S by using a function $f(S)$. Definition of the set $\{S^p(c)\}$ (its cardinality is denoted by `Num_S`) composed by all the positive solutions, that is all the solutions for which $f(S^p(c)) > f^{max}(c-1)$ (where $f^{max}(c-1)$ represents the maximum value of f in the preceding cycle).
3. **Updating of** $P(c)$. To compute all the new $P_i(c+1)$ the subsets $\{S_i^p(c)\}^j$ has to be considered, whose cardinalities are denoted by $N_{i,j}(c)$. They are composed of all the solutions in $\{S^p(c)\}$ in which the i-th element is equal to j ($j = 0$ or 1). Next, it is necessary to compute $\bar{f}_{\{S^p(c)\}}$ and $\bar{f}_{\{S_i^p(c)\}^j}$ representing the average function value on the sets $\{S^p(c)\}$ and $\{S_i^p(c)\}^j$, respectively. Let

$$\lambda_i^j = \frac{\bar{f}_{\{S_i^p(c)\}^j}}{\{S^p(c)\}},$$

then it follows:

$$P_i(c+1) = \frac{P_i(c) + \lambda_i^1}{1 + \lambda_i^0 + \lambda_i^1}$$

These three steps are iterated until the stop condition occurs; in our experiments we used a fixed number of cycles (`Max_Cycles`).

A few words are needed about the choice for each S_i in step 1. Following the above–mentioned step 1, Scout generates solutions corresponding to the binary configuration with higher bit probability value. On the basis of the performance of these solutions, the vector P is updated. The updating, in this situation, will increase P_i's having high values and it will decrease P_i's with low values. After a very few cycles, P_i's will have only values 1 or 0 and the solutions will tend to be equal to each other (each solution will correspond to the bit configuration with the highest probability value), leading to the stagnation of the algorithm. To overcome this situation, the step 1 is also driven by a factor E measuring the tendency to explore alternative paths, similar to the mutation operator in evolutionary algorithms. When the probabilistic choosing has been performed, for each S_i, the algorithm generates a random number r. If this number r is less than the parameter value E (what signals the event "To explore alternative path") the chosen S_i value is inverted. In such a way, the algorithm has the possibility to take unlikely decisions, and to explore new solutions.

A review of this algorithm gives, that it offers a degree of flexibility according to the update rule used in step 3. Reasonably modified forms of the update rule give new variants of the standard Scout algorithm. It was tried to find a modification such that the progress of the Scout algorithm mostly resembles the evolutionary progress of a standard genetic algorithm (SGA).

For doing so, the SGA was run on some optimization tasks, and a "Scout" was derived from the population at each step. The value of P_i of this Scout is computed by the relative amount of occurrences of bit 1 at position k within all individuals of the population with respect to the total number of individuals. Then, the modification of each P_i during a generation step was observed, and it was tried to model it by an appropriate update rule. From this, two possible variations of the Scout algorithm were justified empirically.

The first new version, which relates second–order statistics of the average function value gain of the Scouts (consider [3] for a study of the role of second–order statistics in genetic algorithms) and is denoted Scout_M differs from the original version in the following phases:

1. **Generation'**. No change.
2. **Selection'**. Evaluation of each S by using a function $f(S)$. All the generated solutions S of the cycle c (denoted by $\{S(c)\}$) participate in the updating of vector P.
3. **Updating'**. To compute the new $P_i(c+1)$ the subsets $\{S_i^p(c)\}^j$ has to be considered as for the standard Scout algorithm. Next, it is necessary to compute $\bar{f}_{\{S^p(c)\}}$ and $\bar{f}_{\{S_i^p(c)\}^j}$ representing the average function value on the sets $\{S^p(c)\}$ and $\{S_i^p(c)\}^j$, respectively, and $\sqrt{\sigma_{\{S(c)\}}}$ representing the square root of the standard deviation of the function value on the set $\{S(c)\}$. Then, it follows:

$$P_i(c+1) = P_i(c) + \alpha \frac{\sqrt{\bar{f}_{\{S_i^p(c)\}^1} \bar{f}_{\{S^p(c)\}}}}{\sqrt{\sigma_{\{S(c)\}}}}$$

where α represents an algorithm parameter (a real number belonging to the range [0,1]), which can be considered as a "learning rate" for a similar parameter use in the delta rule of neural network learning.

The second new version (denoted Scout_F), which combines the Scout_M approach with the original Scout algorithm, differs from the original version in the following phases:

1. **Generation"**. No change.
2. **Selection"**. Evaluation of each S by using a function $f(S)$. All the generated solutions S of the cycle c (denoted by $\{S(c)\}$) participate in the updating of vector P.
3. **Updating"**. To compute the new $P_i(c+1)$ the subsets $\{S_i^p(c)\}^j$ has to be considered as for the standard Scout algorithm. Next, it is necessary to compute $\bar{f}_{\{S^p(c)\}}$ and $\bar{f}_{\{S_i^p(c)\}^j}$ representing the average function value on the sets $\{S^p(c)\}$ and $\{S_i^p(c)\}^j$, respectively, and $\sqrt{\sigma_{\{S(c)\}}}$ representing the square root of the standard deviation of the function value on the set $\{S(c)\}$. Let

$$\gamma_i^j = \frac{\sqrt{\bar{f}_{\{S_i^p(c)\}^j}\bar{f}_{\{S^p(c)\}}}}{\sqrt{\sigma_{\{S(c)\}}}},$$

then it follows:

$$P_i(c+1) = \frac{P_i(c) + \gamma_i^1}{1 + \gamma_i^0 + \gamma_i^1}.$$

3 Scout Algorithm and Schemata Theorem

In this section, the question is considered, if Scout algorithms (in standard or modified form) perform a search based on schematas. For the application of the Scout algorithm, a probability vector (PV) $P = (P_1, P_2, \ldots, P_n)$ with $P_i \in [0, 1]$ is used. For each algorithm step, a number m of bitstrings is constructed, using the P_i as partial probabilities. After judging the fitness values of the generated bitstrings, P is modified accordingly.

The question is, if this search strategy could be considered as a schematic one. In order to show this, a possible way would be to confirm two properties:

- Intrinsic parallelism, i.e. the algorithm searches within multiple subspaces at once.
- As the algorithm proceeds, the subspaces become smaller, and the rate of trials allocated to above-average subspaces increases proportionally to this above-averageness, if problem dimension increases.

The following model will be used (see figure 1 for an illustration of the twodimensional case): a PV equals a point P within the n-dimensional unit cube. By drawing the parallel surfaces to the cube surfaces, which cut P, the unit cube is divided into subcubes. The essential property of these subcubes is, that their volumina correspond to the probabilities of a certain bitstring generated by a

PV. To give an example, the lower left area (A_{11}) in figure 1 is the probability for obtaining the bitstring $(1,1)$. Thus, the generation of m bitstrings could be considered as a MONTE CARLO measurement of the volumina of the subcubes. This measurement is done by equally distributing points within the unit cube and counting, how often each subcube was hit.

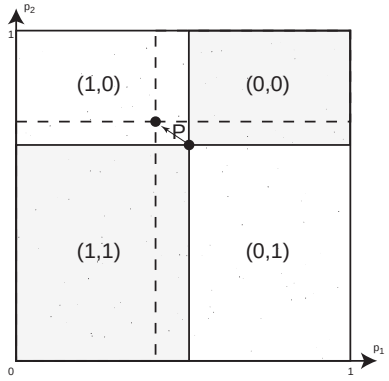

Fig. 1. Probability vector (PV) within the unit square.

From this, the schemata are obvious. They are comprised by the subcubes and by its composites (e.g. the schema $(1\star)$ corresponds to the left half of the unit square, i.e. $A_{11} \cup A_{10}$, in figure 1). Hence, property 1 is verified.

For property 2, the case of displacing P has to be considered. For simplification, we restrict our consideration of the n-dimensional case by assuming $P_1 = P_2 = \ldots = P_n = p$ and $p > 1/2$. P should be displaced by the positive amount δ. Now we ask for given dimension n what size of δ is needed for doubling the ratio r of $A_{11\ldots1}$ to $A_{00\ldots0}$. The old and the new ratios are given by:

$$r_{old} = \left(\frac{p}{1-p}\right)^n$$
$$r_{new} = \left(\frac{p+\delta}{1-(p+\delta)}\right)^n$$

If solving $r_{new}/r_{old} = 2$, we obtain:

$$\delta = \underbrace{\frac{p(1-p)}{1-p+\sqrt[n]{2}}}_{term1} \underbrace{(\sqrt[n]{2}-1)}_{term2}$$

The essential property of this result is that $\sqrt[n]{2} \to 1$ decreases exponentially, if $n \to \infty$ (consider figure 2). For large n, the term 1 becomes constant, and

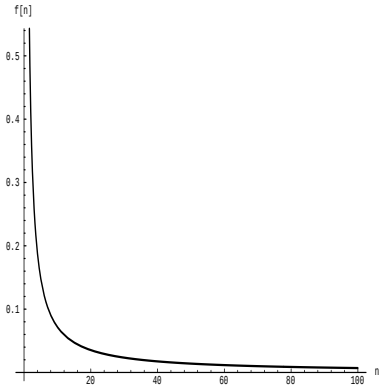

Fig. 2. Plot of $f(n) = \sqrt[n]{2} - 1$.

term 2 quickly approaches Zero. Also, for large n, the decay of term 2 starts to dominate term 1.

The necessary replacement of P in order to double the relative ratio of two subcubes becomes smaller and smaller, as problem dimension increases. The decay is according to an exponential rate. Of course, the same holds for other ratios than 2 as well.

If a schema has order $l < n$, i.e. l positions of the bitstring are clamped, the decay of relative subcube volumina is dominated by a term proportionally to $\sqrt[n-l]{2}$.

Because the rate of allocations to a certain schema equals the volume of the corresponding subcube, moving P towards subcubes with above-average fitness (this is, what the modification law of the Scout algorithm for P actually does) is all what is necessary to fulfill the schemata theorem. Hence, property 2 is also verified.

The Scout algorithm performs a schemata based search.

4 Experimental Validation

In [1], the standard Scout algorithm was tested on the maximum clique problem. In order to relate the performance of the proposed Scout algorithms to the SGA, they were further tested on two well-known problems: the Royal Road Functions (RRF) [7] in the special form presented in [6] as an example of a deceptive problem for SGA, and the Tanese functions [8], which are constructed from Walsh Polynomials (WP), intensively discussed in [2] and which are hard to solve for SGA. To point out similarities of Scout algorithms and SGA, it is expected that the Scout algorithms also give a good performance on the RRF and a poor performance on the Tanese functions (despite of the fact that a modified update rule may improve performance).

4.1 Royal Road Functions Results

In all the experiments on the Royal Road Functions we used the default values, as stated by Holland, for all the input parameters to the RRF.

The parameters used for the three versions of the Scout algorithm were:

- Number of Scouts $K = 10$ for all the versions, $K = 50$, 100 for the original version of the Scout algorithm;
- Exploration parameter E in the range $[0.0, 0.06]$;
- Number of cycles `Max_Cycles` $= 10000$;
- Learning rate $\alpha = 1.0$;
- Number of different runs for each parameter configuration was set to 10; each run started with a different initial situation. The best value found in ten runs is denoted by f_{max}, the average of the best values over ten runs by f_{av}, their standard deviation by σ_f and the cycle of the best value with c_{max}.

The results of the experiments for the standard Scout algorithm are given in table 1. Table 2 gives the results for the Scout_M algorithm and table 3 the results for the Scout_F algorithm.

Table 1. Result of the standard Scout algorithm for the RRF.

K	E	f_{av}	σ_f	f_{max}	c_{max}
10	0.0000	2.204	0.355253	2.620	233
10	0.0500	4.534	0.533837	5.300	4843
10	0.0510	4.636	0.545266	5.500	7744
10	0.0520	4.562	0.539831	5.300	5978
10	0.0530	4.872	0.494970	5.580	4776
10	0.0540	4.662	0.631362	5.660	8107
10	0.0600	4.618	0.587231	5.600	2798
50	0.0000	2.734	0.638335	4.380	46
50	0.0410	6.860	0.450876	7.600	4764
50	0.0430	6.868	0.616672	7.720	6440
50	0.0450	7.200	0.885990	8.540	1783
50	0.0460	6.680	0.518030	8.020	8347
50	0.0480	6.774	0.355159	7.740	4656
50	0.0490	6.468	0.522575	7.600	5044
100	0.0000	2.804	0.613971	3.860	35
100	0.0410	7.872	0.923529	9.620	6972
100	0.0440	7.858	0.755451	9.060	7886
100	0.0450	7.412	0.621196	8.800	6934
100	0.0460	7.802	0.656486	8.860	9624
100	0.0470	7.870	0.841705	8.880	6646
100	0.0490	7.620	0.798109	8.820	7935

Table 2. Result of the Scout_M algorithm for the RRF.

K	E	f_{av}	σ_f	f_{max}	c_{max}
10	0.0000	8.262	5.863806	12.800	24
10	0.0001	12.800	0.000000	12.800	23
10	0.0002	12.800	0.000000	12.800	56
10	0.0003	12.800	0.000000	12.800	59
10	0.0004	12.800	0.000000	12.800	48
10	0.0005	12.800	0.000000	12.800	34
10	0.0010	10.600	4.641839	12.800	34
10	0.0020	10.508	4.832043	12.800	601
10	0.0030	12.800	0.000000	12.800	2790
10	0.0040	4.006	4.655358	12.800	2955
10	0.0050	1.792	0.518391	2.420	3849

Table 3. Result of the Scout_F algorithm for the RRF.

K	E	f_{av}	σ_f	f_{max}	c_{max}
10	0.0000	12.800	0.000000	12.800	19
10	0.0001	12.800	0.000000	12.800	17
10	0.0002	12.800	0.000000	12.800	32
10	0.0003	12.800	0.000000	12.800	18
10	0.0004	12.800	0.000000	12.800	20
10	0.0005	12.800	0.000000	12.800	19
10	0.0010	12.800	0.000000	12.800	34
10	0.0030	12.800	0.000000	12.800	1311
10	0.0050	2.188	0.265865	2.940	2702

4.2 Walsh Polynomials Results

We used the same parameters as Tanese [8] for all the experiments on Walsh Polynomials. We run the three versions of the Scout algorithm on five different randomly generated Walsh polynomials.

The parameters, which were used for the three versions of the Scout algorithm, were:

- Number of Scouts $K = 10$ for all the versions
- Exploration parameter E in the range $[0.0, 0.1]$;
- Number of cycles `Max_Cycles` $= 50000$;

- Learning rate $\alpha = 1.0$;
- The number of different runs for each parameter configuration was set to 10; each run started with a different initial situation.

The results of the experiments are given in table 4; E denotes the exploration parameter, f_{av} the averaged best value on the ten runs, σ_f their standard deviation, f_{max} the best maximum value over the ten runs and c_{max} the number of the cycle where it occurred. The standard Scout algorithm is denoted by S_O, the first modification by S_M and the second modification by S_F.

Table 4. Comparison of the three different update rules of the Scout algorithm on five Walsh polynomials.

	E	f_{av}	σ_f	f_{max}	c_{max}
Walsh Polynomial 1: maximum value 86.11					
S_O	0.098	75.562	3.307707	81.900	37828
S_M	0.021	76.472	3.978143	83.380	43375
S_F	0.015	75.038	3.680018	84.020	11693
Walsh Polynomial 2: maximum value 66.79					
S_O	0.090	59.800	2.261076	64.950	46419
S_M	0.015	60.214	2.635582	66.470	15584
S_F	0.034	60.538	2.648760	66.110	24816
Walsh Polynomial 3: maximum value 79.81					
S_O	0.099	67.252	3.336.351	74.950	18976
S_M	0.014	68.868	3.994012	76.370	6704
S_F	0.024	70.614	3.790395	78.750	35437
Walsh Polynomial 4: maximum value 79.96					
S_O	0.097	70.350	3.222.770	76.180	47809
S_M	0.018	69.884	3.983274	79.960	40006
S_F	0.034	71.318	3.429279	78.300	1622
Walsh Polynomial 5: maximum value 67.33					
S_O	0.098	60.578	2.689939	66.360	24721
S_M	0.020	60.050	2.557903	65.580	7183
S_F	0.029	60.332	2.428684	66.360	45338

5 Discussion

As can be seen from the tables, the new versions of the Scout algorithm, namely Scout_M and Scout_F, outperform the original algorithm. In the experiments on RRF, the new versions approach the global optimum value (12.8) in a few cycles and they require a very small number of evaluations (number of scouts times cycles needed to reach the absolute best value). The original Scout algorithm is

not able to reach the global optimum value, even if the number of evaluations is increased by a factor of 10. In the case of Walsh polynomials, all the three algorithms do not find the global optimum value for the 5 polynomials, but the two modified versions aproach better values than the original Scout algorithm. This second problem is a hard to solve problem for SGA and it comes out to be difficult for the Scout algorithms, too.

What can be seen from the test results is that the Scout algorithms (in either version) perform in a similar manner as SGA. For the Walsh Polynomials, Tanese reported a poor performance. SGA was not able to find the maximum of the 8th order WP, and rarely found an optimum for the 4th order WP. On the other hand, the RRF of Jones [6] supply building-blocks, and deceives hill-climbing algorithms at the same time. The performance of SGA was reported to be good. For the "SGA-like" update rules of the Scout algorithm, the performance increases dramatically, too.

The discussions in sections 1 and 3 and the results in section 4 support the hypothesis of a similarity of Scout algorithms and SGA. Evidence has been given on three levels:

1. The Scout algorithms fulfill basic requirements of procedures, which are capable of admissible detector configuaration and goal attainement. Thus, they have to be qualified as algorithms of the same algorithm familiy, to which SGA belongs.
2. The Schemata theorem applies to Scout algorithms. This was shown in section 3.
3. The Scout algorithms were tested on functions, for which the performance of SGA is well known (RRF for a deceptive problem, which favours SGA; Tanese function as a hard to solve problem for SGA). The more the update rule resembles the "hidden" SGA update rule, the more the Scout algorithm performs like a SGA.

But nevertheless, the Scout algorithm is in some points different from SGA. There are no direct manipulations of bitstrings possible in the algorithm. But there is an update rule, which has no counterpart for the SGA. As it is not possible to directly modify the SGA's inherent "hidden Scout," this is possible for the Scout algorithm. By using a modified update rule, the performance of a Scout algorithm can become better than the performance of the SGA for certain optimization problems (as it was demonstrated on the Tanese functions in section 4).

From this discussion, the essential role of Scout algorithms for algorithm design and for obtaining theoretical insights into evolutionary algorithms becomes more clear.

Future work will concentrate on the mathematical issues raised in the foregoing discussion (e.g. relating update rule and schematas more precisely) and will study algorithms, which combines SGA and Scout algorithms by a manner of "recruiting" Scouts by bitstring individuals (i.e. a mixed population).

References

1. Abbattista, F., Dalbis, D., *The Scout Algorithm to Explore Unknown Spaces*, Proc. of the Int. Conference on Evolutionary Computation, ICEC'98, 1998.
2. Forrest, S., Mitchell, M., *The Performance of Genetic Algorithms on Walsh Polynomials: Some Anomalous Results and Their Explanation*, Proc. of the 4th Int. Conference on Genetic Algorithms, San Diego, CA, Morgan Kaufmann Publishers, San Mateo, CA, 1991.
3. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.
4. Holland, J.H., *Goal–Directed Pattern Recognition*, Proc. of the Int. Conference on Methodologies of Pattern Recognition, Honolulu, Hawaii, pp. 287–296, 1969.
5. Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
6. Jones, T., *A Description of Holland's Royal Road Function*, Evolutionary Computation, 2(3) pp. 409-415, 1995.
7. Mitchell, M., Forrest, S., Holland, J.H., *The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance*, Proc. of the 1st European Conference on Artificial Life, Cambridge, MA, MIT Press, 1991.
8. Tanese, R., *Distributed Genetic Algorithms for Function Optimization*, PhD thesis, The University of Michigan, Ann Arbor, MI, 1989.

This article was processed using the LaTeX macro package with LLNCS style