Robust and Efficient Genetic Algorithms with Hierarchical Niching and a Sustainable Evolutionary Computation Model

Jianjun Hu[1], Erik Goodman[2]

[1,2]Genetic Algorithm Research and Application Group(GARAGe)
[1]Department of Computer Science and Engineering
[2]Department of Electrical and Computer Engineering
Michigan State University, East Lansing, MI, 48823
{Hujianju, Goodman}@egr.msu.edu

Abstract. This paper proposes a new niching method named hierarchical niching, which combines spatial niching in search space and a continuous temporal niching concept. The method is naturally implemented as a new genetic algorithm, QHFC, under a sustainable evolutionary computation model: the Hierarchical Fair Competition (HFC) Model. By combining the benefits of the temporally continuing search capability of HFC and this spatial niching capability, QHFC is able to achieve much better performance than deterministic crowding and restricted tournament selection in terms of robustness, efficiency, and scalability, simultaneously, as demonstrated using three massively multi-modal benchmark problems. HFC-based genetic algorithms with hierarchical niching seem to be very promising for solving difficult real-world problems.

1. Introduction

Genetic algorithms are widely applied to challenging engineering problems today. However, there are still several undesirable properties with current genetic algorithms. The first one is the lack of a quality guarantee of genetic search. For example, genetic algorithms are usually sensitive to the population size in terms of their search capability. Unfortunately, it is difficult to estimate the required population size, despite the extant population sizing theory [1,2]. Too large a population size leads to low efficiency, and one that is too small may simply fail to achieve satisfactory results. The second undesirable property is that once a genetic algorithm stagnates during a search, it usually loses most of its search capability, and there is no good way to rejuvenate the run in an efficient manner. Simple restart or strong mutations may waste the computations spent before by destroying the building blocks in the population. Weak mutations may perturb the solutions a little bit, but they cannot incur significant move in search space once the framework of the individual is established. The third problem of current genetic algorithms is the lack of robustness such as large variation of the performance of several runs due to the opportunistic and convergent nature of current genetic algorithms.

In the past three decades, many niching techniques have been proposed, which have greatly improved the scalability and robustness of genetic search for difficult multi-modal problems [3]. However, due to the convergent nature of the current genetic algorithm framework, these niching approaches still meet difficulty in many hard problems. Based on a sustainable evolutionary computation framework and a hierarchical niching mechanism, this paper proposes a new genetic algorithm, named QHFC, which can significantly improve robustness, efficiency and scalability over that of a representative modern niching approach.

The rest of the paper is organized as follows. In section 2, existing commonly used niching techniques including temporal niching and spatial niching are surveyed, and their three inherent difficulties are outlined. Section 3 then presents the ideas of the sustainable evolutionary computation framework of HFC [4,5], which underlies the design of a new genetic algorithm with hierarchical niching, QHFC to be described in Section 4. A set of three well-known genetic algorithm benchmark problems are used to evaluate QHFC in section 5 and the results are compared to genetic algorithms with deterministic crowding and restricted tournament selection in terms of scalability, efficiency, and robustness. A conclusion is then drawn in Section 6 along with future work to be done.

2. Related Work

The basic framework of genetic algorithms was laid down by John Holland in the 1960s, as summarized in his book [6], following the Darwinian evolution theory of natural selection. Most of the early formulations of evolutionary computation employed the principle of survival of the fittest. But it turned out that incautious keeping of the best individuals leads to bad performance, as population diversity is critical to good evolutionary search. The most widely used techniques to maintain diversity today are niching techniques, including many well-known methods—for example, De Jong crowding [7], deterministic crowding [8], fitness sharing [9], sequential niching [10] and restricted tournament selection [11]. Niching is useful for many application cases of genetic algorithms. It can be used to maintain interim sub-solutions to find a single final solution or to find multiple final solutions. It is also widely used as an effective mechanism to form and maintain diversity in genetic algorithms to solve hard problems. Other methods like reducing selection pressure, selection noise and operator disruption do not typically result in a GA with strong niching behavior. Readers are referred to Mahfoud [3] for an excellent and almost exhaustive review of niching methods.

Niching methods can be classified by their underlying mechanisms [3]. According to the fitness functions employed, they can be categorized as single-environment approaches (such as crowding and sharing) and multiple-environment approaches (such as implicit fitness sharing [12], and multi-objective function optimization). Since multi-environment approaches are usually specific to special types of problems, we are only interested in single-environment niching approaches in this paper. According to whether niching is achieved across space or over time, we have spatial niching and temporal niching. The former includes the widely used crowding and sharing, which form and maintain multiple niches within the space of a single population. The latter form and maintain multiple niches over time. Only one temporal niching approach, called sequential niching [10], has received attention in the literature to date.

Many experimental comparisons and analytical analyses have been conducted to evaluate the advantages and disadvantages of existing niching methods [13, 14, 15]. Mahfoud [13] showed that sequential niching is weak on easy problems and also incapable of solving hard problems due to its lack of cooperation of individuals in niches and the increasing difficulty to find remaining optima. Fitness sharing is a widely used approach and is very strong if used with intelligent scaling and appropriate setting of the sharing radius parameters, both of which, however, are difficult to achieve; bad results have therefore been reported [14,15]. An undesirable property of both sequential niching and fitness sharing is that they modify the search landscape and thus may incur false optima and other unexpected search behaviors. It turns out that deterministic crowding is one of the best spatial niching approaches. It is capable and easy-to-use and its performance has been confirmed by several comparative studies [13, 14]. Compared to fitness sharing, deterministic crowding succeeds with smaller subpopulations and can often find global optima for hard problems [13]. Assuming the selection pressure for high-fitness leads to premature convergence, Hutter [16] proposed a Fitness Uniform Selection Scheme (FUSS) to preserve genetic diversity. However, this approach suffers from insufficient selective pressure for exploitation and unbalanced fitness distribution of the search space. More detailed comparison of FUSS and other diversity maintaining mechanisms with HFC framework [4] is described in [5].

However, there are several difficulties in applying genetic algorithms to practical real-world problems, which lead to situations in which current spatial niching approaches tend to fail miserably. The first constraint of using a genetic algorithm is that we can often use only a very limited population size, at least relative to the size that various sizing methods indicate is needed. However, as spatial niching methods work by spreading the population out across much of the search space, and there are a huge number of local optima, an enormous population size is usually needed to achieve a satisfactory search solution. This has been proved by the population sizing theory associated with deterministic crowding [3]. However, too large a population size leads to a large number of evaluations, which is usually undesirable. This dependence on population size is even made worse by the fact that each niche has to be supported by multiple individuals to search effectively around it.

As a result of the limited population size, spatial niching methods normally fail to maintain a stable subpopulation at low-fitness area of the search space. For example, fitness sharing tends to focus on several high-fitness niches during the later stages of search. The consequence of the loss of low-fitness-level search is that the genetic algorithm may lose the chance of discovering some essential building blocks or other beneficial genetic material in later search stages, focusing instead on building blocks discovered during the very limited sampling experiments in the early search stage. The reason is that the increasingly high average fitness of the population makes it almost impossible to maintain effective search niches at very low fitness levels. This principle is can be interpreted in biological terms as the cost of specialization, or adaptation limiting diversification: adaptation to a specific niche (corresponding to high fitness in a genetic algorithm) theoretically constrains a population's ability to subsequently diversify into other niches [17]. It is in this sense that the ordinary genetic algorithm model is convergent. The progress of fitness corresponds to an entrenching process; the more progress a genetic algorithm makes, the less opportunity it has to find radically new, beneficial structures and then possibly better solutions.

Another difficulty of current spatial niching methods is the uneven pace of progress in the various niches in the early stages. It is often the case that some early-discovered niches tend to attract most of the individuals of the population, while other niches with higher fitness do not attract enough individuals to explore their search domains and expose their potential.

To handle the three difficulties mentioned above—the limited population size, loss of exportation capability, and unbalanced pace of progress of different niches—a new niching approach is needed, based on a new evolutionary algorithm model. In the following section, a new niching method, called hierarchical niching, is proposed. It combines the benefits of both spatial niching and temporal niching, and is implemented in a new sustainable evolutionary search model called the Hierarchical Fair Competition (HFC) model.

3. Hierarchical Niching and the HFC Sustainable Evolutionary Search Model

The basic idea of hierarchical niching is to introduce a continuous version of temporal niching together with spatial niching to address the three difficulties outlined in the previous section. Hierarchical niching here refers to a type of niching technique that maintains continuing search at all (absolute) fitness levels, each of which is subject to a spatial niching technique. It is naturally implemented under a sustainable continuing evolutionary computation model, Hierarchical Fair Competition [4,5,18].

HFC employs an assembly-line structure in which subpopulations are hierarchically organized into different fitness levels [4]. Offspring of a given level are exported to higher levels if their fitness qualifies them for migration. The openings that create are filled by individuals imported from lower levels or generated by mutating other individuals of the same level. The bottom level continuously generates raw genetic material to explore for new building blocks, which are eventually exported to higher fitness levels. The motivation of HFC is to maintain effective search at all fitness levels to sustain the search process indefinitely and thus remove the problem of insufficient sampling and limited population size. The continuing search capability of HFC is achieved by ensuring a continuous supply and incorporation of genetic material in a hierarchical manner, and by culturing and maintaining, but continually renewing, populations of individuals of intermediate fitness levels. It also has the effect of reducing the selection pressure within each subpopulation while maintaining the global selection pressure to help ensure exploitation of good genetic material found. When each subpopulation (level) in an HFC algorithm is updated by application of a spatial niching technique, the hierarchical niching is established.

Hierarchical niching handles the three difficulties mentioned in Section 2 as follows. Since the available population size is too limited to accommodate all local optima simultaneously, hierarchical niching resorts to the continuing search at lower fitness levels to ensure sequential identification of useful building blocks. This is different from sequential niching in the fact that hierarchical niching only allows partial import of recently discovered building blocks from lower levels, which promotes recombination of building blocks discovered early and later. This is in sharp contrast to sequential niching. The issue of loss of explorative capability is handled by the HFC model. In HFC, the lowest fitness level can continuingly generate genetic diversity and export good building blocks to upper levels, so the search power of the genetic algorithm is sustained, and it exhibits no tendency to converge. And because of the mixing of late-discovered building blocks and early-discovered building blocks, HFC works much better than other naïve sustainable search strategies like restarting or multiple runs, in which random genetic material essentially just perturbs current individuals by destroying its building blocks rather than discovering new building blocks. The insufficient sampling and unbalanced pace of progress problems are all handled by the continuing search capability of the HFC model, since lower-level search may go on indefinitely if needed.

Based on hierarchical niching and the HFC model, we have developed a genetic algorithm named QHFC (Q means "quick"), which can achieve significant performance improvement compared to a GA employing another state-of-the-art niching technique, deterministic crowding and restricted tournament selection. The spatial niching used in the current version of QHFC is deterministic crowding, so the demonstration illustrates that QHFC can improve significantly on deterministic crowding alone.

4. The QHFC Algorithm with Hierarchical Niching

QHFC algorithm is designed based on the HFC sustainable evolutionary computation model, the hierarchical niching concept, and the adaptive breeding strategy. Like the multi-population implementation of HFC, the whole population is divided into several levels, each accommodating individuals with fitness within a certain fitness range, except in special situations (to be explained in Table 1 at the end). The QHFC algorithm can be viewed as a set of cooperating GA agents, each searching at a different fitness level, from the lowest (base) level to the top level. Hierarchical niching is implemented as follows: the top level works as a generational GA with deterministic crowding; all other levels update as steady-state GAs with deterministic crowding.

Compared to previous HFC genetic algorithms, one of the most important innovations of QHFC is the *adaptive breeding strategy* implemented using potency testing (discussed next). It provides a generic mechanism to maintain automatically the balance of exploration and exploitation. More specifically, it allows the algorithm to search as greedily as possible, so long as the greedy strategy is sustainable. For easy problems, the top level automatically gets more breeding opportunities and the search is very aggressive. For hard problems where sustained diversity is a necessity, lower levels are automatically bred more frequently to provide the needed influx of diverse individuals for higher levels.

Potency here is defined as the capability of a fitness level in HFC to produce offspring with fitness high enough for export to higher HFC levels. This mechanism for maintaining the potency of all but the top level works as follows: starting from the level just below the top level, breeding is conducted successively in each level, moving toward the lower levels, using steady-state breeding methods, while tracking the number of offspring produced that are eligible for *promotion* (migration to the next-higher fitness level). If a given number of promotable offspring are not produced within a specified number of evaluations at a given level, then a "catch-up" procedure is conducted: a specified fraction of that level's individuals is replaced by individuals taken from (and removed from) the next lower level, and *popsize* genetic operations and evaluations are performed. (*popsize* is the size of the population at the receiving fitness level.) Then, in turn, the openings created at the next lower level are immediately filled with individuals removed from the level below that, etc., until, at the lowest level, the openings are filled by new randomly generated individuals. However, except for the further genetic operations and evaluations performed at the level where the "catch-up" procedure was initiated, further genetic operations and evaluations are not performed as part of this “ripple down” filling of openings. This “double loop” procedure assures that each level, before it next breeds, has either recently produced individuals worthy of promotion to the next level or has received new individuals from the next lower level, thus ensuring its potency to export higher-level individuals. This mechanism for sustaining the potency of search does not require evaluating any measure of the distance among genotypes or phenotypes, and could also be applied to GP and other sorts of problems.

The QHFC algorithm is summarized in Table 1 at the end. Compared with HFC-GP [4] and AHFC-GP [5], QHFC has many fewer parameters to specify, and the admission thresholds are automatically adjusted.

5. Experiments

As discussed in Section 2, we are interested in hard problems with a large number of local optima, typically massively multimodal, with deception. These factors can often expose the limitation of current niching methods if used with a conventional evolutionary computation model. Here, three widely-used massively multimodal and/or deceptive GA test problems are used to evaluate the performance of QHFC with hierarchical niching, and the performance is compared to the modern niching methods deterministic crowding [8] and restricted tournament selection [11], whose performances have been deemed excellent by several other researchers [13,14].

The three benchmark problems used here include:

1) f3deceptive: order-3 deceptive problem [19], with problem sizes n=60, 90, 120, 150, 180, 240, 300

   This deceptive function is composed of separable building blocks of order 3 and has one global optimum at 111…1 and a deceptive attractor at 000…0. There are many local optima in the landscape of this function.

2) 6bipolar: order-6 bipolar deceptive problem [19], with problem sizes n=60, 90, 120, 150, 180, 240, 300

   This deceptive function is composed of separable building blocks of order 6 and has one global optimum at 111…1 and a deceptive attractor at 000…0. There are many local optima in the landscape of this function.

3) trap5: order-5 trap problem [19], with problem sizes n=60, 90, 120, 150, 180, 240, 300

   This deceptive function is composed of separable building blocks of order 5 and has one global optimum at 111…1 and a deceptive attractor at 000…0. There are many local optima in the landscape of this function.

We compared QHFC with one generational GA with deterministic crowding (DC), described in [8], and one steady state GA with restricted tournament selection (RTS) [11]. Since it is difficult to find an appropriate scaling factor and niching radius, evaluation of and comparison with fitness sharing is not reported. But since fitness sharing belongs to the same category of spatial niching techniques and also lacks the capability of maintaining low-level search at later evolutionary stages, we expect that hierarchical niching with QHFC could also improve on the effectiveness of fitness sharing in the same way as it improves deterministic crowding demonstrated below.

Three criteria are used to evaluate the performance of the genetic search:

- Scalability: within a given number of functional evaluations (1,000,000), what is the maximum problem size it can solve to optimum in at least 85% (27) of the total 30 runs?
- Efficiency: for the problem sizes that both QHFC and DCGA can solve, what is average number the evaluations needed to find the global optimum?
- Robustness: What is the fitness variation at the end of 1,000,000 evaluations and what is the variance of the number of evaluations needed to find a global optimum when it is possible (for simplicity, we assume that all failed runs will find the global optima in the next evaluation and use 1,000,000 as the needed evaluations to find a global optimum)? How many runs out of 30 have found the global optimum solution?

The experimental parameters are set as follows:

For QHFC, in all the experiments, with different population sizes and different problem sizes, a single set of parameters was used.

L: 5	γ : 0.8	*breedTopFreq*: 2	*detectExportNo*: 2
percentRefill: 0.25	catchupGen: 20	noprogressGen: 2	

All three algorithms were tested with the same set of parameters for all problems and all problem sizes. The population size for all experiments was 500. All experiments were allowed a generous maximum of 1,000,000 evaluations, and each experiment was repeated 30 times. The per-bit mutation probability for the genetic algorithm with DC and RTS was 0.005, and was zero for QHFC. The window size of RTS was 20, and the tournament size was 2. Note that by adaptive potency testing, QHFC achieves adaptive mutation implicitly. The results of the experiments are summarized in Fig. 1: a-f.

Fig. 1 (a)-(c) shows the average number of evaluations to find the global optimum or to fail because of reaching the limit of 1,000,000 evaluations. It is clear that for all three problems, QHFC found the global optimum with the fewest evaluations, the difference being especially significant in the case of large problem sizes. QHFC also won by having the smallest variation in the number of evaluations needed to find the global optimum. Fig. 1(d)-(f) presents the number of successful runs out of 30 for the three tested niching techniques. The RTS method performed the worst, solving well only for problem size 60 for the three problems, and its performance degraded dramatically when the problem size increased. The DC method was better, but also suffered severely from the limit of the population size. When the problem size reached a threshold, the DC performance also degraded dramatically, as shown in (d-f). For the f3deceptive and 6bipolar problems of size 300, DC only achieved a success rate of 50%. None of the DC runs succeeded for trap5 with a problem size of 300. We also find that genetic algorithms with DC and RTS are very sensitive to the mutation rates for the benchmark problems. When we set the bit mutation probability as 0.00005, restricted tournament selection works much better than deterministic crowding, but both are still much worse than QHFC. It is extraordinary that for

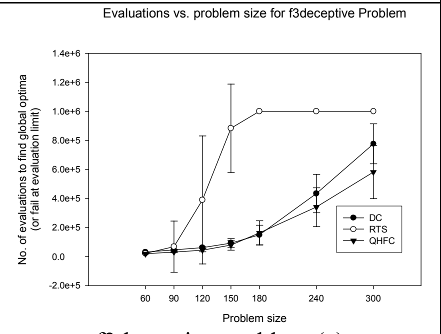
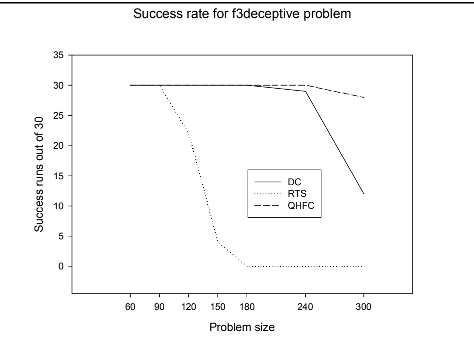
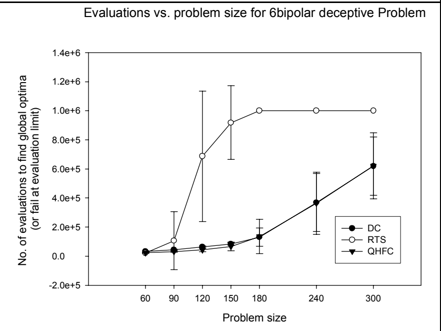
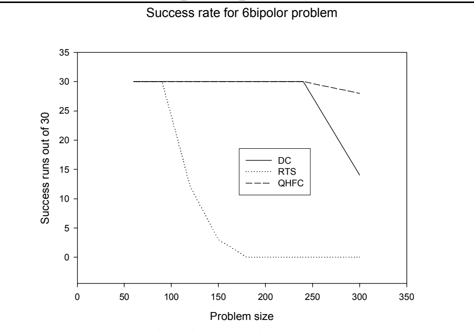
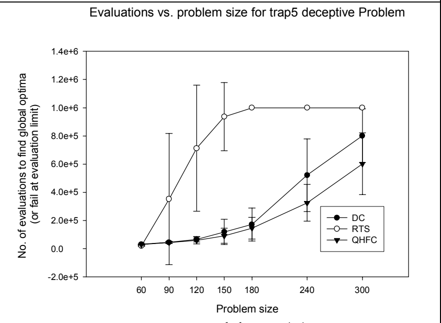
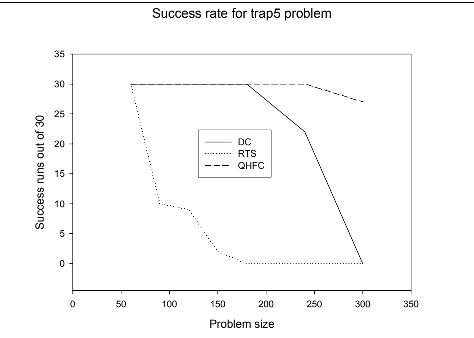
Minimum Evals to find global optimum or fail to find within eval. limit.	Number of Successful runs out of 30
 f3deceptive problem (a)	 f3deceptive problem (d)
 6bipolar problem (b)	 6bipolar problem (e)
 trap5 problem (c)	 trap5 problem (f)

Fig. 1. Comparison of hierarchical niching (QHFC), deterministic crowding (DC), and restricted tournament selection (RTS) in terms of scalability, robustness and efficiency. It is clear that for simple problems or when the problem size is small enough for a population size of 500 is sufficient, DC and RTS work as well as QHFC. However, both DC and RTS suffer from the limited population size and fail for more difficult problems. QHFC clearly has better scalability, robustness, and efficiency.

all three problems, QHFC achieved excellent scalability and solved the problems reliably even at a problem size of 300. Further experiments showed that the performance of QHFC in solving even larger problem sizes degraded very slowly. In fact, experiments (not presented in detail here for lack of space) showed that while DC needed a population size of 4000 to solve a 256-bit HIFF problem [20] with ½ success rate, QHFC solved it reliably with a population size of only 200 for 27 runs out of 30.

We also compared the efficiency of QHFC with the Bayesian Optimization Algorithm (BOA) [19]. For the f3deceptive problem of size 180, BOA needed 160,000 evaluations, while QHFC took an average of 162,717 for 30 runs. For the 6bipolar problem of size 180, BOA took 150,000 evolutions while QHFC needed, on average, 134,966 evaluations. For the trap5 problem of size 180, BOA required 220,000 evaluations, while QHFC took only 145,700 evaluations. Remember that QHFC uses the simple two-point crossover, while BOA explicitly learns the building blocks in these decomposable benchmark problems. It is clear that for decomposable problems with tight building blocks, QHFC with simple crossover is very competitive with BOA. However, BOA works for arbitrary ordering of the variables, in which case the 2-point crossover used in QHFC simply fails, even with the help of the hierarchical niching of QHFC. This demonstrates that the design of representation, linkage learning, and operator design are critical to effective genetic search. It also suggests the possibility of enhancing of BOA-type methods with a hierarchical niching mechanism.

6. Discussion and Conclusions

Robustness, efficiency and scalability are among the most desirable qualities of genetic algorithms. This paper proposed a genetic algorithm, QHFC, which can significantly improve these three performance criteria without significant additional computing effort. The proposed hierarchical niching technique combines the ideas of spatial niching and temporal niching to avoid the pitfalls of insufficient sampling, limited population size, and loss of low-level search capability, all of which contribute to the limited search capability of spatial niching techniques. It should be pointed out that hierarchical niching—the idea of implementing spatial niching at each level of the HFC model—is very different from another temporal niching method, sequential niching. The former achieves good search by promoting the cooperation of niched individuals in all levels, while in the latter method, individuals in the previous run stage cannot help and usually hinder the discovery of later solutions. Compared to spatial niching, hierarchical niching here does not lose the search capability at low fitness levels, while spatial niching methods such as fitness sharing and deterministic crowding are strongly limited by population size and eventually lose search capability at low fitness levels. Another feature of hierarchical niching is that the niching technique used within each level could easily be some method other than the deterministic crowding used in this paper.

The significant performance gain in terms of search sustainability, efficiency, and robustness of QHFC again demonstrates the usefulness of hierarchical niching and of the hierarchical fair competition (HFC) model for sustainable evolutionary search. These algorithms seem to be especially useful for large-scale long-term artificial evolution experiments such as topologically opened synthesis of electric circuits, mechatronic systems, etc.

Our future work will include an experimental comparison study of QHFC with FUSS [16] and fitness sharing with different parameter configurations such as the population sizes. Although our previous work [21] shows that depending on large population size to maintain diversity is not a scalable solution to premature convergence problem, more experiments with more test problems would be helpful to further justify this hypothesis.

References

1 Goldberg, D.E: Sizing Populations for Serial and Parallel Genetic Algorithms, in J.D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Kaufmann, San Mateo, Calif. (1989)

2 Harik, G. R. and Lobo, F.G.: A parameter-less genetic algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference (1999).

3 Mahfoud, S.W.: Niching Methods for Genetic Algorithms, Ph.D. Thesis, University of Illinois at Urbana-Champaign. (1995).

4 Hu, J., Goodman, E.D.: Hierarchical Fair Competition Model for Parallel Evolutionary Algorithms. In *Proceedings, Congress on Evolutionary Computation, CEC 2002*, IEEE World Congress on Computational Intelligence, Honolulu, Hawaii, May. (2002).

5 Hu, J., Goodman, E. D. and Seo, K.: Continuous Hierarchical Fair Competition Model for Sustainable Innovation in Genetic Programming. In *Genetic Programming Theory and Practice*, Kluwer, (2003), pp. 81-98.

6 Holland, J.H.: *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press. (1975).

7 De Jong, K.A.: An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International*, 36(10),514B. (University Microfilms No. 76-9381). (1975).

8 Mahfoud, S. W. Crowding and preselection revisited. In *Proc. Parallel problem Solving from Nature, PPSN '92*, Brussels, (1992).

9 Goldberg, D. E. and Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, (1987). pp. 41--49.

10 Beasley, D., Bull, D. R. and R. R. Martin: A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2): (1993) pp. 101--125

11 Harik, G.: Finding multimodal solutions using restricted tournament selection. In *Proceedings of Sixth International Conference on Genetic Algorithms*, (1995).

12 Darwen, P. and Yao, X.: Every niching method has its niche: Fitness sharing and implicit sharing compared. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature-- PPSN IV*, pages 398--407, Berlin, Springer. (1996).

13 Mahfoud, S.M. (1995): A Comparison of Parallel and Sequential Niching Methods. In *Proceedings of Genetic and Evolutionary Computation Conference*, (1995) pp136--143.

14 Sareni, B., Krahenbuhl, L.: Fitness Sharing and Niching Methods revisited. *IEEE Trans. on Evolutionary Computation*, 2(3), September (1998) pp. 97-106

15 Ursem, R.K: When Sharing Fails. In *Proceedings of the Third Congress on Evolutionary Computation* (CEC-2001), (2001)

16 Hutter, M. Fitness Uniform Selection to Preserve Genetic Diversity. In Proceedings of the 2002 Congress on Evolutionary Computation: 783—788 (CEC-2002), Hawaii, (2002)

17 Buckling, A. et al.: Adaptation limits diversification of experimental bacterial populations. *Science*, December 19, 302, (2003) pp.2107-2109.

18 Hu, J., E. D. Goodman, K. Seo, Z. Fan, R. C. Rosenberg: HFC: a Continuing EA Framework for Scalable Evolutionary Synthesis. In *Proceedings of the 2003 AAAI Spring Symposium - Computational Synthesis: From Basic Building Blocks to High Level Functiona*lity, Stanford, California, March, 24-26, (2003) pp. 106-113.

19 Pelikan, M, Goldberg, D.E. & Erick Cantú-Paz, E. BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-99), I, 525-532.

20 Watson, R. Analysis of recombinative algorithms on a nonseparable building block problem. In *Foundations of Genetic Algorithms 6*, W. Martin and W. Spears, Eds., San Mateo, CA: Morgan Kaufmann, (2001), pp. 69-89.

21 Hu, J., Goodman, E., Seo, K., Fan, Z., Rosenberg, R. The Hierarchical Fair Competition (HFC) Framework for Sustainable Evolutionary Algorithms. *Evolutionary Computation*, 13(1), 2005 (To appear)

Table 1 QHFC Genetic Algorithm with Hierarchical Niching

```
Procedure do_potency_testing ( $l$ )
$l$ is the level for potency testing
catchup_evaluation ← 0
exportedIndividual ← 0
while catchup_evaluation < catchupGen*$|P_l|$ and exportedIndividual< detectExportNo
randomly pick two individuals from level $l$
crossover, mutate, and evaluate
if fitness of offspring > $f_{adm}^{l+1}$,
    promote it (them) to level $l+1$ (replacing randomly any but the best individual or other
    individuals just promoted) and call import_from_below to replace its (their) closest
    parent(s)
        exportedIndividual ← exportedIndividual +1
else
        do deterministic crowding with the 4-member family
endif
end while
if  fail to promote detectExportNo individuals
return not success
else
return success
Procedure end

Procedure import_from_below ( l, nImport, victimList )
$l$: the level into which to import new individuals from next lower level
nImport: the number of individuals to import from next lower level
victimList: a list of indices  of individuals which will be replaced by the imported new
individuals
if $l$=0
    randomly generate nImport new individuals and import into (lowest)  level $l$
else
    randomly choose nImport individuals from  level $l-1$ to replace individuals in victimList .
    If victimList is empty, randomly choose victim individual from current level. Put the
    indices of the new immigrant individuals from level $l-1$ into the level $l-1$
    newVictimList, whose openings will eventually be filled with individuals from level
    $l-2$ (this assures the replacement of individuals removed from level $l-1$ )
call import_from_below ( l -1, nImport, newVictimList)
Procedure end
```

Parameters:

Total population size $|P_t|$ ***L***: number of subpopulations (levels) of QHFC

γ : size factor parameter**,** the ratio of higher level archive size w.r.t next lower level archive size $|P_{k-1}| = |P_k| . \gamma$

breedTopFreq: number of generations to breed top level between potency testing of lower levels (via breeding)

detectExportNo: number of individuals from a level that must be promoted for the level to be considered potent

catchupGen: maximum evaluations in any but top level, normalized by level's popsize, for potency test

percentRefill: percentage of this level's popsize to import from next lower level when there is no progress in the top level, or when lower levels fail potency test (do not furnish *detectExportNo* qualified immigrants within specified number of evaluations)

noprogressGen: maximum number of generations without any fitness progress in top level before triggering importing of *percentRefill* individuals from next lower level

QHFC Main procedure

```
1. initialization
     rancomly initialize and evaluate the HFC subpopulations
     calculate the average fitness of the whole population and set it as the admission fitness of
     the bottom level, $f_{\min}$ , which is fixed thereafter
     remove individuals with fitness less than $f_{\min}$ , and equally distribute the rest of
     the individuals among the levels, according to fitness, thereby determining the
     admission threshold of each level
   generate random individuals to fill the openings in each archive
2. while termination_condition is false
    breed the top level for breedTopFreq generations using generational deterministic
    crowding and applying mutation after each crossover
    if no progress on best fitness of the whole population for noprogressGen
    generations, call import_from_below, but ensuring the best individual is not
    replaced if average fitness of top level > 2 $f_{adm}^{L-1}$ - $f_{adm}^{L-2}$ , adjust admission
    thresholds by evenly allocating fitness range to each level:
         $f_{adm}^{k} = f_{\min} + k(f_{\max} - f_{\min}) / L$   for k=0 to $L$-1
      where $f_{adm}^{k}$ is the admission fitness of level k, $f_{\max}$ is the maximum fitness of the
      whole population
   //potency testing
    for each level from L-2 to 0
  call do_potency_testing
     if not succeed
          call import_from_below to replace (at random) percentRefill percentage of the
          current level, breed one generation at this level
       endif
 end for
end while
End Main
```