A Comparison of Parallel and Sequential Niching Methods

Appears in *Proceedings of the Sixth International Conference on Genetic Algorithms*, 136–143, 1995.

Samir W. Mahfoud
LBS Capital Management, Inc.
311 Park Place Blvd., Suite 330
Clearwater, FL 34619

E-mail: sam@lbs.com

A Comparison of Parallel and Sequential Niching Methods

Samir W. Mahfoud
LBS Capital Management, Inc.
311 Park Place Blvd., Suite 330
Clearwater, FL 34619
E-mail: sam@lbs.com

Abstract

Niching methods extend genetic algorithms to domains that require the location of multiple solutions. This study examines and compares four niching methods — sharing, crowding, sequential niching, and parallel hillclimbing. It focuses on the differences between *parallel* and *sequential* niching. The niching methods undergo rigorous testing on optimization and classification problems of increasing difficulty. A niching-based technique is introduced that extends genetic algorithms to classification problems.

1 INTRODUCTION

Niching methods (Mahfoud, 1995) promote the formation and maintenance of stable subpopulations in genetic algorithms (GAs), allowing GAs to extend their problem-solving power to complex domains. This study examines four niching methods and compares their performances on problems in both classification and multimodal function optimization. The problems cover a wide range of difficulty levels. Classification problems are solved via a new niching-based technique.

Parallel niching methods conceptually form and maintain niches simultaneously within a single population — regardless of the number of processors employed. *Sequential niching* methods, on the other hand, locate multiple niches temporally. This study compares two parallel niching methods, *crowding* and *sharing*, to both a sequential niching method and a parallel hillclimber. It illustrates the strengths and the limitations of all four niching methods.

2 SHARING

Sharing (Goldberg & Richardson, 1987) derates each population element's fitness by an amount related to the number of similar individuals in the population. Specifically, an element's *shared fitness*, f', is equal to its prior fitness f divided by its *niche count*. An individual's niche count is the sum of *sharing function* (sh) values between itself and each individual in the population (including itself). The shared fitness of a population element i is given by the following equation:

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{n} sh(d(i,j))} \quad . \tag{1}$$

The sharing function is a function of the distance d between two population elements; it returns a '1' if the elements are identical, a '0' if they cross some threshold of dissimilarity, and an intermediate value for intermediate levels of dissimilarity. The threshold of dissimilarity is specified by a constant, σ_{share}; if the distance between two population elements is greater than or equal to σ_{share}, they do not affect each other's shared fitness. A common sharing function is

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} , & \text{if } d < \sigma_{share} ; \\ 0 , & \text{otherwise} , \end{cases} \tag{2}$$

where α is a constant that regulates the shape of the sharing function. Both genotypic and phenotypic distance measures can be employed; the appropriate choice depends upon the problem being solved.

3 CROWDING

Crowding techniques (De Jong, 1975) insert new elements into the population by replacing similar elements. We employ the *deterministic crowding* (DC) variation because of its niching capabilities (Mahfoud, 1992, 1994). DC works as follows. First it groups all population elements into $n/2$ pairs. Then it crosses all pairs and mutates the offspring. Each offspring competes against one of the parents that produced it. For each pair of offspring, two sets of parent-child tournaments are possible. DC holds the set of tournaments that forces the most similar elements to compete. Like in sharing, similarity can be measured using either genotypic or phenotypic distances. Pseudocode for deterministic crowding is given on the following page.

Deterministic Crowding

(REPEAT for g generations)

DO $n/2$ times:

1. Select 2 parents, p_1 and p_2, randomly, no replacement
2. Cross them, yielding c_1 and c_2
3. Apply mutation / other operators, yielding c_1' and c_2'
4. IF $[d(p_1, c_1') + d(p_2, c_2')] \leq [d(p_1, c_2') + d(p_2, c_1')]$
   - IF $f(c_1') > f(p_1)$ replace p_1 with c_1'
   - IF $f(c_2') > f(p_2)$ replace p_2 with c_2'

   ELSE
   - IF $f(c_2') > f(p_1)$ replace p_1 with c_2'
   - IF $f(c_1') > f(p_2)$ replace p_2 with c_1'

4 PARALLEL HILLCLIMBING

We define a parallel hillclimber that, starting with a randomly generated initial population, forces each element to converge to its nearest attractor. Attractors are defined using a neighborhood operator that is appropriate to the problem being solved. The neighborhood operator may be defined either over phenotypic (variable) space or Hamming (bit) space. Pseudocode for the phenotypic variation is given below.

Parallel Hillclimbing (Phenotypic)

1. Initialize *Step Size*
2. WHILE *Step Size* $\geq \epsilon$
   (a) FOR each population element
   - Randomly pick a starting variable
   - *Change* = TRUE
   - WHILE *Change*
     - *Change* = FALSE
     - FOR each variable
       - IF adding *Step Size* to current variable yields improved fitness
         - Perform the addition
         - *Change* = TRUE
       - ELSE IF subtracting *Step Size* from current variable yields improved fitness
         - Perform the subtraction
         - *Change* = TRUE

   (b) *Step Size* = *Step Size* / 2

The parallel hillclimbing algorithm is similar in operation to binary search. When operating in phenotypic space, the hillclimber starts with a large *step size*, and each population element hillclimbs until it can no longer improve. The hillclimber then cuts its step size in half, and each population element again hillclimbs until it can no longer improve. The hillclimber iterates until it has hillclimbed using a step size of ϵ, where ϵ is the smallest possible increment in a variable. The phenotypic neighborhood operator simply consists of either an addition or a subtraction of the step size from one of the problem's variables.

The initial step size requires some care in setting. Too small a size will delay the hillclimber's convergence. Too large a size, on the other hand, will cause points to skip from local optimum to local optimum. While such behavior may be satisfactory when the goal is to locate the single best optimum, to locate multiple optima it is preferable that each point remain within the basin of attraction (of the optimum) in which it currently resides. This latter strategy prevents repetitive location of the same few optima. For genotypic hillclimbing, also called *bitclimbing*, we set both the initial and final step sizes to one bit.

Both phenotypic and genotypic hillclimbers implement versions of *next-ascent* hillclimbing (Mühlenbein, 1991). Starting with a randomly chosen variable, they cycle through the variables, trying perturbations on each one. For the phenotypic hillclimber, a perturbation is either an upward or a downward change in the value of a variable. For the genotypic hillclimber, a perturbation is the flipping of a bit-variable. The hillclimbers take each improvement they find. They terminate, for each step size, after they have completed a full cycle through the variables without improvement.

The parallel hillclimber is an important base for comparison, because if a niching method is merely shuffling points and then converging to the nearest attractor, there is no point in pursuing that niching method — parallel hillclimbing is superior. We instead seek niching methods that intelligently decide which maxima to pursue, and that tend to prefer higher maxima over lower maxima. Note that the hillclimbing method we have chosen may not be the most efficient of all possible hillclimbers. However, it has predictable behavior: it tends to climb to the nearest local optimum in whose basin of attraction it currently lies, a desirable property for a hillclimber that strives to locate multiple optima. Also to its merit, our hillclimber is general-purpose, requiring only the same problem information as the GA.

5 SEQUENTIAL NICHING

The sequential niching (SN) method we consider is that of Beasley, Bull, and Martin (1993). It works by iterating a simple GA, and maintaining the best solution of each run off-line. The authors call the multiple runs that sequential niching performs to solve a single problem, a *sequence*. To avoid converging to the same area of the search space multiple times, whenever SN locates a solution, it depresses the fitness landscape at all points within some radius of that solution. This *niche radius* plays a role in SN similar to that of σ_{share} in sharing. In fact, the authors suggest that SN is a sequentialization of fitness sharing.

Like the parallel hillclimber, sequential niching is an important base to which parallel niching methods can be compared. If a parallel niching method can offer no advantage, there is no point in wasting time with it — one would be better off locating solutions sequentially. As we soon demonstrate, the advantages of parallel niching methods go beyond aesthetics: parallel niching can accomplish many things that sequential niching can not. Beasley et al. mention three potential advantages of sequential niching. The first is simplicity: SN is conceptually a simple add-on to existing optimization methods. The second is the ability to work with smaller populations, since the goal during each run of a sequence is to locate only one peak. The third is speed, and is partially a byproduct of the second. We find that the latter two potential advantages never materialize. In fact, many disadvantages quickly become apparent. These include the following:

- Loss, through deration, of optimal solutions and their building blocks;
- Repeated search of depressed regions of the space;
- Repeated convergence to the same solutions;
- Loss of cooperative population properties, including cooperative problem solving, and niche maintenance along the way to a single solution;
- Slower runtime, even on serial machines.

6 METHODOLOGY

We employ a single performance criterion that is common to all algorithms. We assume that all algorithms, given a sufficiently large population, will be able to solve all problems that we consider. This is not an unrealistic assumption, since the probability of all desired solutions appearing in the initial population approaches one as population size approaches infinity. Our performance criterion is the total number of function evaluations for the minimum population size at which an algorithm returns all desired solutions. (We do not penalize or reward an algorithm for returning extraneous solutions.) Over multiple trials, the fewer the average number of function evaluations, the better an algorithm is for solving a particular problem.

We start with the minimum population size that is both a power of two and large enough to locate all desired solutions ($n = 2$ for SN). We double n and rerun an algorithm until either it locates and maintains to termination all desired solutions, or it exceeds a function-evaluations limit. The limit is 1.5 million GA function evaluations or, alternatively, 2 million combined function evaluations for the GA plus the hillclimber. (This combination is explained shortly.) We use only population sizes that are powers of two in order to avoid massive numbers of trials and fine-grained distinctions between nearly optimal population sizes. For SN, like Beasley et al., we employ a constant population size across all runs in a sequence.

SN and sharing run under stochastic universal selection (SUS) (Baker, 1987). We employ SUS because it is the least noisy of commonly used, unbiased, fitness-proportionate selection methods. DC and parallel hillclimbing have built-in selection mechanisms.

After each GA terminates, including GAs internal to SN sequences, hillclimbing is invoked upon that GA's final population. The idea is to deal only with local optima and not points in the neighborhoods of local optima. GAs are, after all, global optimization methods, and are well complemented by local optimization methods such as hillclimbers. We use the parallel hillclimbing algorithm as our post-GA hillclimber. (There is no need, of course, to call the parallel hillclimber to optimize its own final population.) For sequential niching (and for the other three algorithms as well), hillclimbing occurs on the original, un-derated, fitness landscape. This is one of Beasley et al.'s suggested improvements to their algorithm.

We employ full crossover (with probability 1.0) and no mutation in all three genetic algorithms. Without mutation, we are better able to determine the merits and drawbacks of the underlying GAs.

All four niching algorithms return all unique elements in the final population (post-hillclimbing) as solutions. For SN, when a single run of a sequence locates multiple solutions, these solutions are added to the final list. SN then derates about each solution, as if multiple runs had been performed. The final population for SN consists of the combination of final populations (post-hillclimbing) of all runs in a sequence.

We run each algorithm 10 times on each test problem, employing the same 10 random number seeds (and initial populations) for each algorithm. We terminate each run (including runs internal to SN sequences) after it effectively stops improving. (Parallel hillclimbing is an exception; it runs until no further improvement occurs.) We determine the stopping point in a fashion similar to Beasley et al., by employing a halting window of five generations. Call the current generation that has just finished t_0, and the prior four generations, t_{-1}, t_{-2}, t_{-3}, and t_{-4}. If the average fitness of the population at t_0 is not more than some increment *inc* greater than the average fitness of the population at t_{-4}, the run halts. We use $inc = .001$ on all problems except *M6*, where we use $inc = .1$. Runs of an SN sequence halt when the average *derated* fitness of the population stagnates. Sharing and DC halt when the average *raw* fitness stagnates.

The stopping criterion for the overall SN algorithm is not easy to set, at least when the algorithm is successful. We give SN ideal behavior by stopping successful sequences immediately after they have located all de-

sired peaks. Unsuccessful sequences are stopped after some run fails to yield a new solution, or after they exceed the maximum number of function evaluations.

We utilize the sharing function of Equation 2 with $\alpha = 2$. Beasley et al. recommend $\alpha = 2$ instead of the usual $\alpha = 1$, in order to produce lower false optima in the derated fitness landscape. Like Beasley et al., we set the derating function for sequential niching to one minus the equivalent sharing function.

7 TEST PROBLEMS

We consider 11 problems, ranging widely in difficulty. *M1–M9* are multimodal function optimization problems; *MUX-6* and *PAR-8* are classification problems.

The first four problems *M1–M4* are one-dimensional, five-peaked, sinusoidal functions. Similar functions were first used by Goldberg and Richardson (1987). Despite the simplicity of these functions, most potential niching GAs have in the past had trouble locating and maintaining all five peaks. *M1* (shown in Figure 1) consists of equally-spaced peaks of uniform height; *M2*, equally-spaced peaks of nonuniform height; *M3*, unequally-spaced peaks of uniform height; *M4*, unequally-spaced peaks of nonuniform height. The variable x in *M1–M4* is encoded using 30 bits. The functions are specified by the following equations:

$$
\begin{aligned}
M1(x) &= \sin^6(5\pi x) \\
M2(x) &= e^{-2(\ln 2)(\frac{x-0.1}{0.8})^2} \sin^6(5\pi x) \\
M3(x) &= \sin^6(5\pi[x^{0.75} - .05]) \\
M4(x) &= e^{-2(\ln 2)(\frac{x-0.08}{0.854})^2} \sin^6(5\pi[x^{0.75} - .05])
\end{aligned}
$$

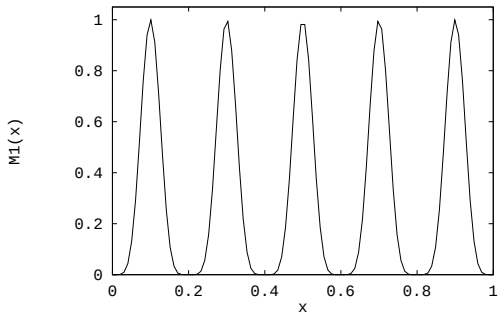

Figure 1: Test Function *M1* is displayed.

The fifth problem *M5*, shown in Figure 2, is the modified Himmelblau's function from Deb's (1989) study. *M5* is a two-dimensional function with four peaks of identical height. The two variables, x and y, are encoded using 15 bits apiece. *M5* is defined below.

$$M5(x,y) = \frac{2186 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2}{2186}$$

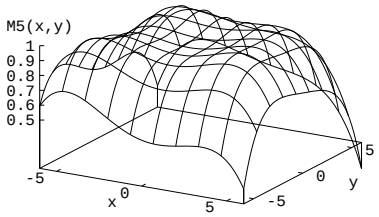

Figure 2: Test Function *M5* is displayed.

M6, shown in Figure 3, is the Shekel's Foxholes problem from De Jong's (1975) dissertation. *M6* is a two-dimensional function with twenty-five peaks of differing heights. The two variables are encoded using 17 bits apiece. Let $a(i) = 16[(i \bmod 5) - 2]$ and $b(i) = 16(\lfloor i/5 \rfloor - 2)$. *M6* is defined below.

$$M6(x,y) = 500 - \frac{1}{.002 + \sum_{i=0}^{24} \frac{1}{1+i+(x-a(i))^6+(y-b(i))^6}}$$

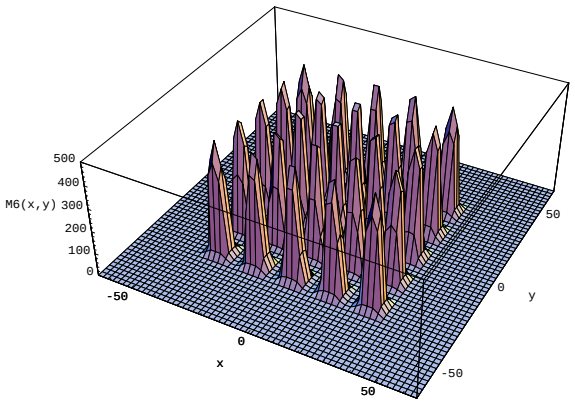

Figure 3: Test Function *M6* is displayed.

M7 is the massively multimodal, deceptive function of Goldberg, Deb, and Horn (1992). Overall fitness is the sum of the fitnesses of five subfunctions. Each subfunction is a bimodal, deceptive function of unitation, as displayed in Figure 4. The total number of optima is $5,153,664$, of which 32 are global. *M8* is the same as *M7*, but exponentially scaled to create larger differentials between the fitnesses of global and nonglobal optima. The scaling function is

$$M8 = 5(\frac{M7}{5})^{15} \quad ,$$

where *M7* in the equation represents the value for *M7* over the entire 30-bit function.

M9 is a minimum-distance function (Horn & Goldberg, in press). Overall fitness is the sum of the fitnesses of three subfunctions of eight bits each. Each

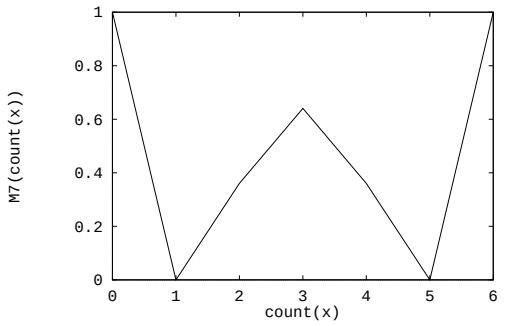

Figure 4: Test Function *M7* is displayed.

subfunction is a maximally deceptive, trimodal function. Global optima are arbitrarily chosen at points, 00000000, 10001100, and 01001010. The fitness of a substring is its Hamming distance to the closest global, except for global substrings, which receive a fitness of 10. *M9* has 2197 local optima, of which 27 are global.

The remaining two problems are classification problems. Given a set of positive and negative training examples, the objective is to find a concept description that includes all of the positive examples, but no negative examples. We map this to a multimodal optimization problem by using the full population as a disjunctive-normal-form concept description, and by letting each population element represent a disjunct. The niching GA must locate and maintain a set of optimal disjuncts. We assign fitnesses to individual disjuncts based on the number of positive examples they cover (POS), and the number of negative examples they cover (NEG). Assuming NTX total negative examples, our fitness function is

$$f(POS, NEG) = \begin{cases} 1 + POS & \text{if } NEG = 0 \ ; \\ 1 - \frac{NEG}{NTX} & \text{otherwise} \ . \end{cases}$$

Individuals in the population are represented by a concatenation of two-bit values that represent boolean variables. A '00' value corresponds to a '0' bit; '11', to a '1' bit; '01' and '10', to a wild card (a "don't care" symbol). A repair mechanism reduces the size of the search space. The repair mechanism flips all '10' alleles, upon fitness assignment, to '01' alleles.

We examine two types of boolean classification problems that are extensively used in the machine-learning literature — parity problems and multiplexer problems. Parity problems form a class of boolean problems that are maximally hard in a sense: from an optimization point of view, they contain the largest numbers of optima of any boolean concept. Multiplexer problems represent average-case problems.

PAR-8 is an eight-bit, odd-parity problem: if an odd number of variables are "on", the example is positive; otherwise, the example is negative. *PAR-8* requires the location and maintenance of 128 disjuncts. *MUX-6* is a six-bit multiplexer problem. A multiplexer contains a number of address bits and a number of data bits. The address bits determine the data bit that is selected.

We employ the following niche radius for each problem. This value serves as the niche radius for sequential niching, σ_{share} for sharing, and the initial step size for parallel hillclimbing, both on its own and when tacked on to the end of a GA. *M1–M4* use a phenotypic niche radius of .1; *M5*, 4.24 (phenotypic); *M6*, 8.0 (phenotypic); *M7* and *M8*, 5.5 (genotypic); *M9*, 2.5 (genotypic); *MUX-6* and *PAR-8*, 3.5 (genotypic). These values allow discrimination among desired peaks.

8 RESULTS

Tables 1–3 summarize all results and allow comparison of the four algorithms on problems of varying difficulty. The tables compare the number of GA function evaluations (without hillclimbing function evaluations added) for the three GAs. They then compare the total number of function evaluations (GAs plus hillclimbers) for all four algorithms. The best average results on each problem are shown in boldface.

We group test problems based on difficulty. The first of three groups, shown in Table 1, consists of the easiest problems, *M1–M5* and *MUX-6*. Each of these problems has fewer than ten peaks and has negligible isolation or misleadingness. The second group, shown in Table 2, consists of two problems of intermediate difficulty, *M6* and *PAR-8*. These problems have a moderate number of peaks — *M6* has 25 and *PAR-8* has 128 — and have negligible isolation or misleadingness. The third group of problems, shown in Table 3, consists of the three hardest problems, *M7–M9*. Each of these problems has thousands to millions of peaks, and also displays both isolation and misleadingness.

In Tables 1–3, average subpopulation size is computed by dividing average population size by the number of desirable peaks. For sequential niching, average population size is first computed across all sequences and all runs within a sequence. This average population size is then divided by the number of desirable peaks to yield average subpopulation size. For SN, $\bar{g}$ indicates the average, across all sequences, of the combined number of generations for all runs within a sequence.

Table 1 shows that on the easiest problems, parallel hillclimbing is the overall winner, with DC a close second. Parallel hillclimbing is fastest on four of the six easy test functions, and DC is fastest on the other two. Second place goes twice to hillclimbing, twice to DC, and twice to sharing. Third place goes four times to sharing, once to DC, and once to SN. SN takes last place all but one time: it edges out DC on *M5*.

These results confirm prior results of GA researchers who show that hillclimbing algorithms can outperform

Table 1: Performances of parallel hillclimbing (HC), sequential niching (SN), fitness sharing (SH), and deterministic crowding (DC) are given on the six easiest test functions. Statistics are taken over ten runs. Average subpopulation size is $\bar{n}$; average number of generations is $\bar{g}$. The mean number of function evaluations (μ) is given for each GA alone, and for each combination of GA and hillclimber. The best average results on each problem are shown in boldface.

Method	$\bar{n}$	$\bar{g}$	*GA:* μ	*Combo:* μ
		M1		
HC	2.72			**1017**
SN	3.68	46.40	738	4112
SH	5.76	8.00	**264**	2431
DC	2.40	28.00	380	1246
		M2		
HC	2.72			**1021**
SN	4.64	75.60	1770	8632
SH	8.96	8.70	442	3827
DC	2.40	27.40	**372**	1264
		M3		
HC	3.04			1150
SN	5.92	26.80	719	4375
SH	6.08	8.70	294	2579
DC	2.08	20.30	**262**	**1013**
		M4		
HC	3.04			1140
SN	5.12	72.40	2445	10231
SH	6.72	9.20	352	2892
DC	2.08	17.00	**210**	**975**
		M5		
HC	2.50			**901**
SN	1.30	32.30	180	1456
SH	2.80	8.00	**103**	1111
DC	5.60	25.60	603	2459
		MUX-6		
HC	10.40			**1257**
SN	6.40	140.70	4423	9439
SH	13.60	8.90	**534**	1931
DC	12.00	44.30	2816	3654

GAs on easy test functions, such as the five functions of De Jong's (1975) test suite. (Note that DC, a crossover hillclimber, is a close second and sometimes defeats the parallel hillclimber.) There is hence no point in doing performance comparisons involving GAs, exclusively using easy problems. One should be more interested in how performance scales up to harder problems. Doing comparisons on a range of problems from very easy to very hard tells a lot about the merits of an algorithm and the scalability of its performance.

We give results without hillclimbing to show how long it takes a GA to get close to the final desired solutions (anywhere within their hillclimbing basins of attraction), without having to pinpoint them. When hillclimbing function evaluations are not counted, sharing and deterministic crowding share three victories apiece, with sharing having the lowest overall number of function evaluations. Second place goes to sharing three times, DC twice, and SN once. Third and final place goes to SN five times and DC once. (There is no fourth place, since we do not compare the hillclimber to the GAs in terms of GA function evaluations.)

It is somewhat surprising that SN performs poorly in comparison with the other algorithms on the six easiest test functions, especially since Beasley et al. employ *M1–M5* in their study, and since our SN results without hillclimbing are in all cases faster than Beasley et al.'s on *M1–M5*. Ignoring hillclimbing, SN requires roughly 3–8 times as many evaluations as sharing, on five of the six easy test functions, and roughly 1.7 times as many on the other. With hillclimbing, SN requires roughly 4–9 times as many function evaluations as parallel hillclimbing on five out of six functions, and roughly 1.6 times as many on the other.

A plausible explanation for SN's behavior is that once SN has squashed several peaks in the fitness landscape, locating the final peak is harder because that peak is isolated. One observation about SN on *M1–M4* is that once its population grows large enough to locate one peak, it has grown large enough to locate multiple peaks. Many runs on *M1–M4* locate no peaks with $n = 8$, but 2–3 peaks at a time with $n = 16$.

All of the algorithms, including SN, find all desired optima on all six test functions in under 11,000 function evaluations. Therefore, all four algorithms are general-purpose enough to handle the easiest problems. We already know that the GA, although a capable optimizer of easy functions, may not be the optimal algorithm for any particular easy test function. A good algorithm, however, must scale up to solving harder problems.

Table 2 illustrates the performances of the four algorithms on two functions of intermediate complexity, *M6* and *PAR-8*. Sharing is the clear winner on both functions, with or without hillclimbing. Sharing performs only 14% to 44% of the function evaluations of its closest competitor. On *M6*, hillclimbing comes in second and SN comes in third. DC fails to locate and maintain the 25 optima, even given 1.5 million GA function evaluations. However, it consistently locates and maintains the global optimum. On *PAR-8*, DC comes in second, taking roughly three times as many function evaluations as sharing. Hillclimbing comes in third, taking about four times as many as sharing. SN comes in fourth, taking about five times as many as sharing. Without hillclimbing, SN edges out DC for second, using 80% of DC's function evaluations.

The reason for DC's peculiar performance on *M6* is that the function is not multimodal in crossover-hillclimbing space. In other words, DC uses the non-global optima as stepping stones to the global optimum: they are all on the crossover path to the global. In terms of the crossover interactions defined in Mahfoud's (1994) study, the global optimum is dominating

Table 2: Performances are given on the two functions of intermediate difficulty.

Method	$\bar{n}$	$\bar{g}$	*GA:* μ	*Combo:* μ
M6				
HC	12.29			29,017
SN	3.58	146.30	12,202	46,657
SH	5.12	11.80	**1,638**	**12,910**
DC			$> 1.5 \times 10^6$	
PAR-8				
HC	48.08			202,387
SN	19.20	36.40	100,557	263,666
SH	9.60	12.60	**17,203**	**54,402**
DC	11.20	87.40	125,850	149,022

Table 3: Performances are given on the three functions of greatest difficulty. Function evaluations are in thousands (indicated by the letter K).

Method	$\bar{n}$	$\bar{g}$	*GA:* μ	*Combo:* μ
M7				
HC				$>$ 2000K
SN			$>$ 1500K	
SH			$>$ 1500K	
DC	20.80	119.80	**81K**	**101K**
M8				
HC				$>$ 2000K
SN			$>$ 1500K	
SH	19.20	19.20	**13K**	**38K**
DC	22.40	134.40	98K	119K
M9				
HC				$>$ 2000K
SN			$>$ 1500K	
SH				$>$ 2000K
DC	136.53	337.80	**1253K**	**1342K**

nearby locals that are in turn dominating other locals, and so on, creating a cascading effect.

The method by which SN solves *PAR-8* is of interest: SN locates all 128 disjuncts in a single run. Its success is therefore due to SUS's stability and not to iterating the GA. Again, once n is high enough to find all desired solutions, it is high enough to find them in one run of a parallel niching method. Unfortunately, this ability of SUS to maintain multiple solutions of identical fitness is not consistent from problem to problem, and evaporates when the solutions have differing fitnesses.

On the three hardest problems (see Table 3), DC is the only method to solve all three in the allotted number of function evaluations. Sharing solves *M8* in fewer evaluations than DC, but reaches the limit on *M7* and *M9*. SN and parallel hillclimbing fail to find the required optima in 1.5 million GA function evaluations and 2 million overall function evaluations, respectively.

Results for sharing on *M7* and *M8* are consistent with those reported by Goldberg, Deb, and Horn (1992), where sharing is unable to solve the unscaled, massively multimodal, deceptive problem, but is able to solve the scaled version. The trouble is that sharing without scaling is too generous in allocating individuals to the millions of extraneous peaks; sharing with scaling minimizes the heights of those peaks. The authors estimate that a population size of over 3.46 million would be necessary to solve the unscaled version. Population sizing formulas for sharing (Mahfoud, in press) give a higher required population size of over 95 million. Sharing encounters problems with *M9* for the same reasons it has trouble with *M7* — too many extraneous peaks of too high a fitness. With $n = 32,768$, sharing converges after 19 generations, performing 655,360 GA function evaluations and 2,227,120 total function evaluations. It returns 2009 total optima, of which 5 are global.

Parallel hillclimbing fails on all three problems for obvious reasons: the problems have become too complex for hillclimbing. The misleading attractors, which correspond to extraneous peaks, draw most population elements that are not exact instances of global optima.

On *M7* and *M8*, SN has the choice of trying to squash millions of extraneous peaks and then trying to converge to 32 remaining needles (in a huge haystack), or trying to locate the globals one at a time. Derating millions of undesirable optima is not a very appealing option. With sufficiently large populations, SN's first few runs locate several global optima. However, after SN derates these and other local optima, the algorithm has a harder time locating any more global optima. This problem becomes progressively worse until the algorithm fails. SN runs into similar problems on *M9*.

Overall, on the 11 test problems, sharing shows the greatest stability of the tested algorithms, typically exhibiting lower standard deviation in the average number of function evaluations to convergence. Sharing also typically runs the fewest generations. DC succeeds with the smallest subpopulations on the average.

9 DISCUSSION OF RESULTS

We can draw the following general conclusions about algorithmic performance versus problem hardness:

- Parallel hillclimbing is best for the easiest problems. It may also work in a reasonable time-frame on problems of intermediate complexity. However, it fails on problems of high complexity.
- Sequential niching is weak on easy problems, and is unable to solve harder problems. In general, parallel hillclimbing is a better method that is also parallel. Parallel hillclimbing outperforms sequential niching because the parallel hillclimber does not destroy the fitness landscape.
- Sharing generally works on problems of all levels of complexity. However, it runs into trouble on problems in which many extraneous peaks exist

that are similar in fitness to the desired peaks. It may be able to overcome this difficulty through the intelligent application of fitness scaling.

- Deterministic crowding is generally good for problems of all levels of complexity. However, it may ultimately lose lower optima that lie on a crossover path to higher optima. Deterministic crowding, a crossover hillclimber, can solve problems that are much more difficult than those solvable by traditional, mutation-based hillclimbing.

We have found that parallel niching methods outperform sequential niching methods. Furthermore, SN does not achieve a sequentialization of fitness sharing, and it yields few of the benefits that its authors claim. In general, parallel niching methods offer the following advantages over sequential niching methods:

- Parallel niching methods can easily be implemented on parallel machines. Sequential niching methods, by their nature, can not.
- Parallel niching methods are faster than SN methods (on serial machines) and give better results.
- Parallel niching GAs can be applied to maintain internal diversity when searching for a single solution. SN can not.
- SN is likely to locate the same solutions repeatedly, despite its deration of peaks.
- SN creates false optima in the derated fitness landscape that are in close proximity to peaks that were previously located. SN can also offset an optimum's location as a consequence of deration. Beasley et al. suggest hillclimbing in the original fitness landscape to overcome both problems. However, hillclimbing stands a good chance of rediscovering a previously derated peak.
- For classification and simulation problems, parallel niching allows the population to cooperatively act as a solution. Sequential niching does not. For instance, one might not be interested in global optima at any point in time but in the state of the population as a whole — the optimal population rather than the optimal population element.
- SN's deration of optima may delete other optima of interest within the deration neighborhood. Even worse, solutions that have been derated — whether local optima, global optima, or near-optima — might take with them important building blocks for locating other solutions. On hard problems such as *M7–M9*, eliminating one global optimum hinders the location of others.
- As SN derates optima, the remaining optima become increasingly difficult to locate. Derated regions, containing mostly plateaus and small ridges, occupy a greater and greater portion of the space — and SN must repeatedly search through this derated portion of the space. After deration of a few optima, the required population size for SN can easily exceed that for a parallel niching method capable of locating all niches within its single population. Beasley et al.'s claim that SN requires a population of only $1/c$ of the size of a parallel niching method's population does not hold, except possibly for locating the first peak.

We have also found that parallel niching GAs outperform parallel hillclimbers, on all but the easiest problems. When many extraneous attractors are present or extraneous attractors with large basins are present, parallel hillclimbing will in probability converge to several of these attractors. GAs with parallel niching, on the other hand, have the power to escape these attractors and to converge to the desired solutions.

References

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Proc. 2nd ICGA*, 14–21.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation, 1*(2), 101–125.

Deb, K. (1989). *Genetic algorithms in multimodal function optimization* (Masters thesis / TCGA Rep. 89002). U. of Alabama, The Clearinghouse for Genetic Algorithms.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems (Doctoral dissertation, U. of Michigan). *Dissertation Abstracts International, 36*(10), 5140B. (University Microfilms No. 76-9381)

Goldberg, D. E., Deb, K., & Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. In R. Männer & B. Manderick (Eds.), *Parallel problem solving from nature, 2* (pp. 37–46). Elsevier.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proc. 2nd ICGA*, 41–49.

Horn, J., & Goldberg, D. E. (in press). Genetic algorithm difficulty and the modality of fitness landscapes. In L. D. Whitley (Ed.), *Foundations of genetic algorithms [FOGA], 3*. Morgan Kaufmann.

Mahfoud, S. W. (1992). Crowding and preselection revisited. In R. Männer & B. Manderick (Eds.), *Parallel problem solving from nature, 2* (pp. 27–36). Elsevier.

Mahfoud, S. W. (1994). Crossover interactions among niches. *Proc. 1st IEEE Conf. Evolut. Comput.*, 188–193.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms* (Doctoral dissertation / IlliGAL Rep. 95001). Urbana: U. of Illinois, Illinois Genetic Algorithms Lab.

Mahfoud, S. W. (in press). Population size and genetic drift in fitness sharing. In L. D. Whitley (Ed.), *FOGA, 3*. Morgan Kaufmann.

Mühlenbein, H. (1991). Evolution in time and space — The parallel genetic algorithm. In G. J. E. Rawlins (Ed.), *FOGA* (pp. 316–337). Morgan Kaufmann.