The Crowding Approach to Niching in Genetic Algorithms

Ole J. Mengshoel omengshoel@riacs.edu
RIACS, NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035, USA

David E. Goldberg deg@uiuc.edu
Illinois Genetic Algorithms Laboratory, Department of General Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract

A wide range of niching techniques have been investigated in evolutionary and genetic algorithms. In this article, we focus on niching using crowding techniques in the context of what we call local tournament algorithms. In addition to deterministic and probabilistic crowding, the family of local tournament algorithms includes the Metropolis algorithm, simulated annealing, restricted tournament selection, and parallel recombinative simulated annealing. We describe an algorithmic and analytical framework which is applicable to a wide range of crowding algorithms. As an example of utilizing this framework, we present and analyze the probabilistic crowding niching algorithm. Like the closely related deterministic crowding approach, probabilistic crowding is fast, simple, and requires no parameters beyond those of classical genetic algorithms. In probabilistic crowding, subpopulations are maintained reliably, and we show that it is possible to analyze and predict how this maintenance takes place. We also provide novel results for deterministic crowding, show how different crowding replacement rules can be combined in portfolios, and discuss population sizing. Our analysis is backed up by experiments that further increase the understanding of probabilistic crowding.

Keywords

Genetic algorithms, niching, crowding, deterministic crowding, probabilistic crowding, local tournaments, population sizing, portfolios.

1 Introduction

Niching algorithms and techniques constitute an important research area in genetic and evolutionary computation. The two main objectives of niching algorithms are (i) to converge to multiple, highly fit, and significantly different solutions, and (ii) to slow down convergence in cases where only one solution is required. A wide range of niching approaches have been investigated, including sharing (Goldberg and Richardson, 1987; Goldberg et al., 1992; Darwen and Yao, 1996; Pétrowski, 1996; Mengshoel and Wilkins, 1998), crowding (DeJong, 1975; Mahfoud, 1995; Harik, 1995; Mengshoel and Goldberg, 1999; Ando et al., 2005b), clustering (Yin, 1993; Hocaoglu and Sanderson, 1997; Ando et al., 2005a), and other approaches (Goldberg and Wang, 1998). Our main focus here is on crowding, and in particular we take as a starting point the crowding approach known as deterministic crowding (Mahfoud, 1995). Strengths of deterministic crowding are that it is simple, fast, and requires no parameters in addition to those of a classical

GA. Deterministic crowding has also been found to work well on test functions as well as in applications.

In this article, we present an algorithmic framework that supports different crowding algorithms, including different replacement rules and the use of multiple replacement rules in portfolios. While our main emphasis is on the *probabilistic crowding* algorithm (Mengshoel and Goldberg, 1999; Mengshoel, 1999), we also investigate other approaches including deterministic crowding. As the name suggests, probabilistic crowding is closely related to deterministic crowding, and as such shares many of deterministic crowding's strong characteristics. The main difference is the use of a probabilistic rather than a deterministic replacement rule (or acceptance function). In probabilistic crowding, stronger individuals do not always win over weaker individuals, they win proportionally according to their fitness. Using a probabilistic acceptance function is shown to give stable, predictable convergence that approximates the niching rule, a gold standard for niching algorithms.

We also present here a framework for analyzing crowding algorithms. We consider performance at equilibrium and during convergence to equilibrium. Further, we introduce a novel portfolio mechanism and discuss the benefit of integrating different replacement rules by means of this mechanism. In particular, we show the advantage of integrating deterministic and probabilistic crowding when selection pressure under probabilistic crowding only is low. Our analysis, which includes population sizing results, is backed up by experiments that confirm our analytical results and further increase the understanding of how crowding and in particular probabilistic crowding operates.

A final contribution of this article is to identify a class of algorithms to which both deterministic and probabilistic crowding belongs, *local tournament algorithms*. Other members of this class include the Metropolis algorithm (Metropolis et al., 1953), simulated annealing (Kirkpatrick et al., 1983), restricted tournament selection (Harik, 1995), elitist recombination (Thierens and Goldberg, 1994), and parallel recombinative simulated annealing (Mahfoud and Goldberg, 1995). Common to these algorithms is that competition is localized in that it occurs between genotypically similar individuals. It turns out that slight variations in how tournaments are set up and take place are crucial to whether one obtains a niching algorithm or not. This class of algorithms is interesting because it is very efficient and can easily be applied in different settings, for example by changing or combining the replacement rules.

We believe this work is significant for several reasons. As already mentioned, niching algorithms reduce the effect of premature convergence or improve search for multiple optima. Finding multiple optima is useful, for example, in situations where there is uncertainty about the fitness function and robustness with respect to inaccuracies in the fitness function is desired. Niching and crowding algorithms also play a fundamental role in multi-objective optimization algorithms (Fonseca and Fleming, 1993; Deb, 2001) as well as in estimation of distribution algorithms (Pelikan and Goldberg, 2001; Sastry et al., 2005). We enable further progress in these areas by explicitly stating new and existing algorithms in an overarching framework, thereby improving the understanding of the crowding approach to niching. There are several informative experiments that compare different niching and crowding GAs (Ando et al., 2005b; Singh and Deb, 2006). However, less effort has been devoted to increasing the understanding of crowding from an analytical point of view, as we do here. Analytically, we also make a contribution with our portfolio framework, which enables easy combination of different replacement rules. Finally, while our focus is here on discrete multimodal optimization,

a variant of probabilistic crowding has successfully been applied to hard multimodal optimization problems in high-dimensional continuous spaces (Ballester and Carter, 2003, 2004, 2006). We hope the present work will act as a catalyst to further progress in this area.

The rest of this article is organized as follows. Section 2 presents fundamental concepts. Section 3 discusses local tournament algorithms. Section 4 discusses our crowding algorithms and replacement rules, including the probabilistic and deterministic crowding replacement rules. In Section 5, we analyze several variants of probabilistic and deterministic crowding. In Section 6, we introduce and analyze our approach to integrating different crowding replacement rules in a portfolio. Section 7 discusses how our analysis compares to previous analysis, using Markov chains, of stochastic search algorithms including genetic algorithms. Section 8 contains experiments that shed further light on probabilistic crowding, suggesting that it works well and in line with our analysis. Section 9 concludes and points out directions for future research.

2 Preliminaries

To simplify the exposition we focus on GAs using binary strings, or bitstrings $\boldsymbol{x} \in \{0,1\}^m$, of length m. Distance is measured using Hamming distance $\textsc{Distance}(\boldsymbol{x}, \boldsymbol{y})$ between two bitstrings, $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^m$. More formally, we have the following definition.

DEFINITION 1 (Distance): *Let $\boldsymbol{x}, \boldsymbol{y}$ be bitstrings of length m and let, for $x_i \in \boldsymbol{x}$ and $y_i \in \boldsymbol{y}$ where $1 \leq i \leq m$, $d(x_i, y_i) = 0$ if $x_i = y_i$, $d(x_i, y_i) = 1$ otherwise. Now, the distance function* $\textsc{Distance}(\boldsymbol{x}, \boldsymbol{y})$ *is defined as follows:*

$$\textsc{Distance}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} d(x_i, y_i).$$

Our DISTANCE definition is often called genotypic distance; when we discuss distance in this article the above definition is generally assumed.

A natural way to analyze a stochastic search algorithm's operation on a problem instance is to use discrete time Markov chains with discrete state spaces.

DEFINITION 2 (Markov chain): *A (discrete time, discrete state space) Markov chain $\mathcal{M}$ is defined as a 3-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$ where $\mathcal{S} = \{s_1, \ldots, s_k\}$ defines the set of k states while $\mathcal{V} = (\pi_1, \ldots, \pi_k)$, a k-dimensional vector, defines the initial probability distribution. The conditional state transition probabilities $\mathcal{P}$ can be characterized by means of a $k \times k$ matrix.*

Only time-homogenous Markov chains, where $\mathcal{P}$ is constant, will be considered in this article. The performance of stochastic search algorithms, both evolutionary algorithms and stochastic local search algorithms, can be formalized using Markov chains (Goldberg and Segrest, 1987; Nix and Vose, 1992; Harik et al., 1997; De Jong and Spears, 1997; Spears and De Jong, 1997; Cantu-Paz, 2000; Hoos, 2002; Moey and Rowe, 2004a,b; Mengshoel, 2008). Unfortunately, if one attempts exact analysis, the size of $\mathcal{M}$ becomes very large even for relatively small problem instances (Nix and Vose, 1992; Mengshoel, 2008). In Section 7 we provide further discussion of how our analysis provides an approximation compared to previous exact Markov chain analysis results.

In $\mathcal{M}$, some states $\mathcal{O} \subset \mathcal{S}$ are of particular interest since they represent globally optimal states, and we introduce the following definition.

DEFINITION 3 (Optimal states): *Let $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$ be a Markov chain. Further, assume a fitness function $f : \mathcal{S} \to \mathbb{R}$ and a globally optimal fitness function value $f^* \in \mathbb{R}$ that defines globally optimal states $\mathcal{O} = \{s \mid s \in \mathcal{S} \text{ and } f(s) = f^*\}$.*

The fitness function f and the optimal states $\mathcal{O}$ are independent of the stochastic search algorithm and its parameters. In general, of course, neither $\mathcal{M}$ nor $\mathcal{O}$ are explicitly specified. Rather, they are induced by the fitness function, the stochastic search algorithm, and the search algorithm's parameter settings. Finding $s^* \in \mathcal{O}$ is often the purpose of computation, as it is given only implicitly by the optimal fitness function value $f^* \in \mathbb{R}$. More generally, we want to not only find globally optimal states, but also locally optimal states $\mathcal{L}$, with $\mathcal{O} \subseteq \mathcal{L}$. Finding locally optimal states is, in particular, the purpose of niching algorithms including crowding GAs. Without loss of generality we consider maximization problems in this article; in other words we seek global and local maxima in a fitness function f.

3 Crowding and Local Tournament Algorithms

In traditional GAs, mutation and recombination is done first, and then selection (or replacement) is performed second, without regard to distance (or the degree of similarity) between individuals. Other algorithms, such as probabilistic crowding (Mengshoel and Goldberg, 1999; Mengshoel, 1999), deterministic crowding (Mahfoud, 1995), parallel recombinative simulated annealing (Mahfoud and Goldberg, 1995), restricted tournament selection (Harik, 1995), the Metropolis algorithm (Metropolis et al., 1953), and simulated annealing (Kirkpatrick et al., 1983) operate similar to each other and different from traditional GAs. Unfortunately, this distinction has not always been clearly expressed in the literature. What these algorithms, which we will here call local tournament algorithms, have in common is that the combined effect of mutation, recombination, and replacement creates local tournaments; tournaments where distance plays a key role. In some cases this happens because the operations are tightly integrated, in other cases it happens because of explicit search for similar individuals. Intuitively, such algorithms have the potential to result in niching through local tournaments: Similar individuals compete for spots in the population, and fit individuals replace those that are less fit, at least probabilistically. The exact nature of the local tournament depends on the algorithm, and is a crucial factor in deciding whether we get a niching algorithm or not. For instance, elitist recombination (Thierens and Goldberg, 1994) is a local tournament algorithm, but it is typically not considered a niching algorithm.

An early local tournament algorithm is the Metropolis algorithm, which originated in physics, and specifically in the area of Monte Carlo simulation for statistical physics (Metropolis et al., 1953). The Metropolis algorithm was later generalized by Hastings (Hastings, 1970), and consists of generation and acceptance steps (Neal, 1993). In the generation step, a new state (or individual) is generated from an existing state; in the acceptance step, the new state is accepted or rejected with a probability following an acceptance probability distribution. Two common acceptance probability distributions

are the Metropolis and the Boltzmann distributions. The Boltzmann distribution is

$$\Pr(E_j) = \frac{\exp(-E_j/T)}{\exp(-E_j/T) + \exp(-E_i/T)} \tag{1}$$

where E_i and E_j are the energies of the old and new states (individuals), respectively, and T is temperature.

Simulated annealing is essentially the Metropolis algorithm with temperature variation added. Variation of the temperature T changes the probability of accepting a higher-energy state (less fit individual). At high temperature, this probability is high, but it decreases with temperature. Simulated annealing consists of iterating the Metropolis algorithm at successively lower temperatures, and in this way it finds an estimate of the global optimum (Kirkpatrick et al., 1983; Laarhoven and Aarts, 1987). Both the Metropolis rule and the Boltzmann rule achieve the Boltzmann distribution

$$\Pr(E_i) = \frac{\exp(-E_i/T)}{\sum_j \exp(-E_j/T)} \tag{2}$$

where $\Pr(E_i)$ is the probability of having a state i with energy E_i at equilibrium; T is temperature. If cooling is slow enough, simulated annealing is guaranteed to find an optimum. Further discussion is provided in Section 4.3.

Within the field of GAs proper, an early local tournament approach is preselection. Cavicchio introduced preselection, in which a child replaces an inferior parent (Goldberg, 1989). DeJong turned preselection into crowding (DeJong, 1975). In crowding, an individual is compared to a randomly drawn subpopulation of c members, and the most similar member among the c is replaced. Good results with $c = 2$ and $c = 3$ were reported by DeJong on multimodal functions (DeJong, 1975).

In order to integrate simulated annealing and GAs, the notion of Boltzmann tournament selection was introduced (Goldberg, 1990). Two motivations for Boltzmann tournament selection were asymptotic convergence (as in simulated annealing) and providing a niching mechanism. The Boltzmann (or logistic) acceptance rule, shown in Equation (1), was used. Boltzmann tournament selection was the basis for parallel recombinative simulated annealing (PRSA) (Mahfoud and Goldberg, 1995). PRSA also used Boltzmann acceptance, and introduced the following two rules for handling children and parents: (i) In double acceptance and rejection, both parents compete against both children. (ii) In single acceptance and rejection, each parent competes against a predetermined child in two distinct competitions. Like simulated annealing, PRSA uses a cooling schedule. Both mutation and crossover are used, to guarantee convergence to the Boltzmann distribution at equilibrium. Three different variants of PRSA were tested empirically with good results, two of which have proofs of global convergence (Mahfoud and Goldberg, 1995). Deterministic crowding (Mahfoud, 1995) is similar to PRSA. Differences are that deterministic crowding matches up parents and children by minimizing a distance measure over all parent-child combinations, and it uses the deterministic acceptance rule of always picking the best fit individual in each parent and child pair.

Another local tournament algorithm is the gene-invariant GA (GIGA) (Culberson, 1992). In GIGA, children replace the parents (Culberson, 1992). Parents are selected, a family constructed, children selected, and parents replaced. Family construction

Table 1: Two Key Dimensions of Local Tournament Algorithms: (i) the Nature of the Acceptance (or Replacement) Rule and (ii) the Nature of the Current State of the Algorithm's Search Process

	Population-based	**Non-population-based**
Probabilistic acceptance	Probabilistic crowding Parallel recombinative simulated annealing	Metropolis algorithm Simulated annealing Stochastic local search
Deterministic acceptance	Deterministic crowding Restricted tournament selection	Local search

amounts to creating a set of pairs of children, and from this set one pair is picked according to some criterion, such as highest average fitness or highest maximal fitness. The genetic invariance principle is that the distribution over any one position on the gene does not change over time. GIGA with no mutation obeys the genetic invariance principle, so the genetic material of the initial population is retained. In addition to selection pressure provided by selection of better child pairs in a family, there is selection pressure due to a sorting[1] effect in the population combined with selection of adjacent individuals in the population array.

Restricted tournament selection is another local tournament algorithm (Harik, 1995). The approach is a modification of standard tournament selection, based on local competition. Two individuals $\boldsymbol{x}$ and $\boldsymbol{y}$ are picked, and crossover and mutation is performed in the standard way, creating new individuals $\boldsymbol{x}'$ and $\boldsymbol{y}'$. Then w individuals are randomly chosen from the population, and among these the closest one to $\boldsymbol{x}'$, namely $\boldsymbol{x}''$, competes with $\boldsymbol{x}'$ for a spot in the new population. A similar procedure is applied to $\boldsymbol{y}'$. The parameter w is called the window size. The window size is set to be a multiple of s, the number of peaks to be found: $w = c \times s$, where c is a constant.

In summary, important dimensions of local tournament algorithms are the form of the acceptance rule, whether the algorithm is population-based, whether temperature is used, which operators are used, and whether the algorithm gives niching or not. Table 1 shows two of the key dimensions of local tournament algorithms, and how different algorithms are classified along these two dimensions. The importance of the distinction between probabilistic and deterministic acceptance is as follows. In some sense, and as discussed further in Section 5, it is easier to maintain a diverse population with probabilistic acceptance, and such diversity maintenance is the goal of niching algorithms. Processes similar to probabilistic acceptance occur elsewhere in nature, for instance in chemical reactions and in statistical mechanics.

Algorithmically, one important distinction concerns how similar individuals are brought together to compete in the local tournament. We distinguish between two approaches. The *implicit* approach, of which PRSA, deterministic crowding, and probabilistic crowding are examples, integrates the operations of variation and selection to set up local tournaments between similar individuals. The *explicit* approach, examples of which are crowding and restricted tournament selection, searches for similar

[1]Culberson's approach induces a sorting of the population due to the way in which the two children replace the two parents: The best fit child is placed in the population array cell with the highest index. Better fit individuals thus gradually move toward higher indexes; worse fit individuals toward lower indexes.

individuals in the population in order to set up local tournaments. Restricted tournament selection illustrates that local tournament algorithms only need to have their operations conceptually integrated; the key point is that individuals compete locally (with similar individuals) for a spot in the population. So in addition to variation and selection, the explicit approach employs an explicit search step.[2] Whether a local tournament algorithm gives niching or not depends on the nature of the local (family) tournament. If the tournament is based on minimizing distance, the result is niching, else no niching is obtained. For example, deterministic crowding, restricted tournament selection, and probabilistic crowding are niching algorithms, while elitist recombination and GIGA are not.

The focus in the rest of this article is on the crowding approach to niching in evolutionary algorithms. While our main emphasis will be on probabilistic crowding and deterministic crowding, the algorithmic and analytical frameworks presented are more general and can easily be applied to other crowding approaches.

4 Crowding in Genetic Algorithms

Algorithmically, we identify three components of crowding, namely crowding's main loop (Section 4.1); the transition or step from one generation to the next (Section 4.2); and finally the issue of replacement rules (Section 4.3). A number of replacement rules are discussed in this section; our main focus is on the ProbabilisticReplacement and DeterministicReplacement rules.

4.1 The Main Loop

The main loop of our CrowdingGA is shown in Figure 1. Without loss of generality, we assume that CrowdingGA's input fitness function f is to be maximized. Initialize initializes each individual in the population. Then, for each iteration of the **while**-loop in the CrowdingGA, local tournaments are held in order to fill up the population array newPop, based on the existing (old) population array oldPop. Occupation of one array position in newPop is decided through a local tournament between two or more individuals, where each individual has a certain probability of winning. Tournaments are held until all positions in the population array have been filled. The CrowdingGA delegates the work of holding tournaments to either SimpleStep, which is presented in Figure 2, or the CrowdingStep, which is presented in Figure 3. As reflected in its name, SimpleStep is a simple algorithm that is—in certain cases—amendable to exact analysis. The CrowdingStep algorithm, on the other hand, is more general but also more difficult to analyze. In this section we focus on the algorithmic aspects, while in Section 5 we provide analysis.

4.2 Stepping Through the Generations

Two different ways of going from one generation to the next are now presented, namely the SimpleStep algorithm and the CrowdingStep algorithm.

[2]Note that explicit versus implicit is a matter of degree, since deterministic or probabilistic crowding with crossover perform an optimization step in order to compute parent-child pairs that minimize total distance. This optimization step is implemented in Match in Figure 3.

```
CROWDINGGA(n, S, P_M, P_C, g_N, R, f, general)
 Input:  n         population size
         S         size of family to participate in tournaments
         P_M       probability of mutation
         P_C       probability of crossover
         g_N       number of generations
         R         replacement rule returning true or false
         f         fitness function
         general   true for CROWDINGSTEP, else for SIMPLESTEP false
 Output: newPop    final population of individuals
begin
    g_C ← 0 {Initialize current generation counter}
    oldPop ← NEW(n) {Create population array with n positions}
    newPop ← NEW(n) {Create second population array with n positions}
    INITIALIZE(oldPop) {Typically, initialization is uniformly at random}
    while g_C < g_N
        if (general)
            newPop ← CROWDINGSTEP(oldPop, S, P_M, P_C, g_C, R, f)
        else
            newPop ← SIMPLESTEP(oldPop, P_M, g_C, R, f)
        end
        oldPop ← newPop
        g_C ← g_C + 1
    end
    return newPop
end
```

Figure 1: Pseudo-code for the main loop of our crowding GA. A population array oldPop is used as input to CROWDINGSTEP or SIMPLESTEP, and a new population array newPop is created from it, using also the variation operators as implemented in the respective step algorithms.

```
SIMPLESTEP(oldPoP, PM, g_C, R, f)

 Input:  oldPop    old population of individuals
         P_M       probability of mutation
         g_C       current generation number
         R         replacement rule returning true or false
         f         fitness function
 Output: newPop    new population of individuals
begin
    i ← 1 {Counter variable}
    while i ≤ SIZE(oldPop) {Treat all individuals in the population}
        child ← oldPop[i] {Create child by copying parent in population}
        MUTATE(child, P_M)
        if R(f(parent[i]), f(child), g_C) {Tournament using replacement rule R}
            newPop[i] ← child {Child wins over parent}
        else
            newPop[i] ← oldPop[i] {Parent wins over child}
        end
        i ← i + 1
    end
    return newPop
end
```

Figure 2: Pseudo-code for one step of a simple crowding GA that uses mutation only.

```
CROWDINGSTEP(oldPop, S, P_M, P_C, g_C, R, f)
 Input:  oldPop   population of individuals - before this step
         S        even number of parents (with S ≥ 2) in tournament
         P_M      probability of mutation
         P_C      probability of crossover
         g_C      current generation number
         R        replacement rule returning true or false
         f        fitness function
 Output: newPop   new population of individuals
begin
    k ← 1 {Begin Phase 0: Create running index for newPop}
    for i ← 1 to SIZE(oldPop) step 1
        indexPool[i] ← i
    while SIZE(indexPool) > 1 {Continue while individuals are left in oldPop}
        for i ← 1 to S step 1 {Begin Phase 1: Select parents from oldPop}
            random ← RANDOMINT(1, SIZE(indexPool)) {Uniformly at random}
            j ← indexPool[random]
            parent[i] ← oldPop[j]
            REMOVE(indexPool,random) {Remove index of random individual}
        for i ← 1 to S step 2 {Begin Phase 2: Perform crossover and mutation}
            if P_C ⩾ RANDOMDOUBLE(0, 1) then {Pick random number in [0, 1]}
                CROSSOVER(parent[i], parent[i + 1], child[i], child[i + 1],P_C)
            else
                child[i] ← parent[i]
                child[i + 1] ← parent[i + 1]
            MUTATE(child[i], P_M)
            MUTATE(child[i + 1], P_M)
        for i ← 1 to S step 1 {Begin Phase 3: Select i-th parent}
            for j ← 1 to S step 1 {Select j-th child}
                distance[i, j] ← DISTANCE(parent[i], child[j])
        m* ← MATCH(distance, parent, child, S) {Phase 4: Compute matchings}
        for i ← 1 to S step 1 {Begin Phase 5: Invoke rule for each m_i ∈ m*}
            c ← child[childIndex(m_i)] {Get index of child in match m_i}
            p ← parent[parentIndex(m_i)] {Get index of parent in match m_i}
            if R(f(p), f(c), g_C) {Tournament using replacement rule R}
                w ← c {Child is winner w in local tournament}
            else
                w ← p {Parent is winner w in local tournament}
            newPop[k] ← w {Put winner w into new population}
            k ← k + 1
    return newPop
end
```

Figure 3: Pseudo-code for one step of a general crowding GA. It is assumed, for simplicity, that popSize is a multiple of the number of parents S. All phases operate on a family consisting of S parents and S children. In Phase 3, distances are computed for all possible parent-child pairs. In Phase 4, matching parent-child pairs are computed, minimizing a distance metric. In Phase 5, tournaments are held by means of a replacement rule. The rule decides, for each matching parent-child pair, which individual wins and is placed in newPop.

4.2.1 A Simple Crowding Approach

The SIMPLESTEP crowding algorithm is presented in Figure 2. The algorithm iteratively takes individuals from oldPop, applies a variation operator MUTATE, and uses a replacement rule R in order to decide whether the parent or child should be placed into the next generation's population newPop. The SIMPLESTEP algorithm is a stepping

stone for CROWDINGSTEP. The relatively straightforward structure of SIMPLESTEP simplifies analysis (see Section 5) and also makes our initial experiments more straightforward (see in particular Section 8.1).

4.2.2 A Comprehensive Crowding Approach

The CROWDINGSTEP algorithm is presented in Figure 3. The CROWDINGSTEP consists of several phases, which we present in the following.

Phase 0 of CROWDINGSTEP: all valid indexes of the population array are placed in the indexPool. The indexPool is then gradually depleted by repeated picking from it without replacement in the following step.

Phase 1 of CROWDINGSTEP: First, parents are selected uniformly at random without replacement. This is done by picking indexes into newPop (using RANDOMINT) and then removing those indexes from the indexPool (using REMOVE). For the special case of $S = 2$, the effect is that the CROWDINGSTEP randomly selects two parents $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ from the population, similar to classical tournament selection.

Phase 2 of CROWDINGSTEP: In this phase, the CROWDINGSTEP performs one-point crossover and bit-wise mutation using the CROSSOVER and MUTATION algorithms, respectively. Two parents are crossed over with probability P_C using CROSSOVER, which creates two children $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. The crossover point is decided inside the CROSSOVER operator. After crossover, a bit-wise MUTATION operator is applied to $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ with probability P_M, creating mutated children $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$.

Phase 3 of CROWDINGSTEP: This is the phase where distances between parents and children in a family are computed. This is done by filling in the distance-array using the DISTANCE algorithm, see Definition 1. In the $S = 2$ special case, distances are computed for all combinations of the two mutated children $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$ with the two parents $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, giving a 2×2 distance array. In general, the size of the distance array is S^2. For the case of $S = 2$, the 2×2 distance array is populated as follows: distance[1, 1] $\leftarrow$ DISTANCE($\boldsymbol{p}_1, \boldsymbol{c}'_1$), distance[1, 2] $\leftarrow$ DISTANCE($\boldsymbol{p}_1, \boldsymbol{c}'_2$), distance[2, 1] $\leftarrow$ DISTANCE($\boldsymbol{p}_2, \boldsymbol{c}'_1$), and distance[2, 2] $\leftarrow$ DISTANCE($\boldsymbol{p}_2, \boldsymbol{c}'_2$).

Phase 4 of CROWDINGSTEP:[3] The algorithm MATCH and the distances computed in Phase 3 are used to compute an optimal matching $\boldsymbol{m}^* = \{m_1, \ldots, m_S\}$, where each match m_i is a 2-tuple containing one parent and one child. For the $S = 2$ case, the matchings considered are

$$\boldsymbol{m}_1 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_1), (\boldsymbol{p}_2, \boldsymbol{c}'_2)\} \tag{3}$$

and

$$\boldsymbol{m}_2 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_2), (\boldsymbol{p}_2, \boldsymbol{c}'_1)\}. \tag{4}$$

The corresponding total distances d_1 and d_2 are defined as follows

$$d_1 = \text{DISTANCE}(\boldsymbol{p}_1, \boldsymbol{c}'_1) + \text{DISTANCE}(\boldsymbol{p}_2, \boldsymbol{c}'_2) = \text{distance}[1,1] + \text{distance}[2,2] \tag{5}$$

$$d_2 = \text{DISTANCE}(\boldsymbol{p}_1, \boldsymbol{c}'_2) + \text{DISTANCE}(\boldsymbol{p}_2, \boldsymbol{c}'_1) = \text{distance}[1,2] + \text{distance}[2,1], \tag{6}$$

[3]Note that other crowding algorithms, which use mutation only, and no crossover, have a lesser need for this matching step. For reasonably small mutation probabilities, one can assume that the child $\boldsymbol{c}$, created from a parent $\boldsymbol{p}$, will likely be very close to $\boldsymbol{p}$.

and determine which matching is returned by MATCH. Continuing the $S = 2$ special case, the output from MATCH is either $\boldsymbol{m}^* = \boldsymbol{m}_1 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_1), (\boldsymbol{p}_2, \boldsymbol{c}'_2)\}$ (3) or $\boldsymbol{m}^* = \boldsymbol{m}_2 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_2), (\boldsymbol{p}_2, \boldsymbol{c}'_1)\}$ (4). MATCH picks $\boldsymbol{m}_1$ (3) if $d_1 < d_2$, else $\boldsymbol{m}_2$ (4) is picked.

Generally, each individual in the population is unique in the worst case, therefore $\boldsymbol{m}^*$ is one among $S!$ possibilities $\boldsymbol{m}_1 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_1), (\boldsymbol{p}_2, \boldsymbol{c}'_2), \ldots, (\boldsymbol{p}_S, \boldsymbol{c}'_S)\}$, $\boldsymbol{m}_2 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_2), (\boldsymbol{p}_2, \boldsymbol{c}'_1), \ldots, (\boldsymbol{p}_S, \boldsymbol{c}'_S)\}, \ldots, \boldsymbol{m}_{S!} = \{(\boldsymbol{p}_1, \boldsymbol{c}'_S), (\boldsymbol{p}_2, \boldsymbol{c}'_{S-1}), \ldots, (\boldsymbol{p}_S, \boldsymbol{c}'_1)\}$. The complexity of a brute-force implementation of MATCH is clearly $S!$ in the worst case. For large S, where the size of $S!$ becomes a concern, one would not take a brute-force approach but instead use an efficient algorithm such as the Hungarian weighted bipartite matching algorithm. This algorithm uses two partite sets, in our case the parents $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_S\}$ and children $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_S\}$, and performs matching in $O(S^3)$ time.

Our minimization of total distance in MATCH is similar to that performed in deterministic crowding (Mahfoud, 1995), and there is a crucial difference to matching in PRSA (Mahfoud and Goldberg, 1995). Using PRSA's single acceptance and rejection replacement rule, each parent competes against a predetermined child rather than against the child that minimizes total distance as given by DISTANCE. In other words, only one of the two matchings $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, say $\boldsymbol{m}_1$, is considered in PRSA.

Phase 5 of CROWDINGSTEP: This is the local tournament phase, where tournament winners $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots\}$ are placed in newPop according to the replacement rule R.[4] More specifically, this phase consists of holding a tournament within each pair in $\boldsymbol{m}^*$. Suppose, in the case of $S = 2$, that the matching $\boldsymbol{m}^* = \boldsymbol{m}_1 = \{(\boldsymbol{p}_1, \boldsymbol{c}'_1), (\boldsymbol{p}_2, \boldsymbol{c}'_2)\}$ is the output of MATCH. In this case, tournaments are held between $\boldsymbol{p}_1$ and $\boldsymbol{c}'_1$ as well as between $\boldsymbol{p}_2$ and $\boldsymbol{c}'_2$, producing two winners $\boldsymbol{w}_1 \in \{\boldsymbol{p}_1, \boldsymbol{c}'_1\}$ and $\boldsymbol{w}_2 \in \{\boldsymbol{p}_2, \boldsymbol{c}'_2\}$. The details of different replacement rules that can be used for R are discussed in Section 4.3.

4.2.3 Discussion

To summarize, we briefly discuss similarities and differences between CROWDINGSTEP and SIMPLESTEP. Clearly, their overall structure is similar: First, one or more parents are selected from the population, then one or more variation operators are applied, and then finally similar individuals compete in local tournaments. In this article, a variation operator is either mutation (used in CROWDINGSTEP and SIMPLESTEP) or crossover (used in CROWDINGSTEP). Similarity, or short distance, between individuals may come about implicitly, as is the case when mutation only is employed in SIMPLESTEP, or explicitly, for instance by minimizing a distance measure in MATCH as part of CROWDINGSTEP or by explicitly searching for similar individuals in the population (Harik, 1995). In all cases, one or more tournaments are held per "step." If $\boldsymbol{p}$ and $\boldsymbol{c}$ are two similar individuals that have been picked to compete, formally $(\boldsymbol{p}, \boldsymbol{c}) \in \boldsymbol{m}^*$, then the result of executing the replacement rule $R(f(\boldsymbol{p})), f(\boldsymbol{c}))$ decides which of $\boldsymbol{p}$ and $\boldsymbol{c}$ is elected the tournament winner $\boldsymbol{w}$ and is placed in the next generation's population newPop by CROWDINGSTEP or SIMPLESTEP. Obviously, there are differences between CROWDINGSTEP and SIMPLESTEP as well: SIMPLESTEP does not include crossover or explicit computation of distances and matchings.

[4] This phase would also be a natural place to include domain or application heuristics, if available, into the crowding algorithm. Such heuristics would be invoked before the replacement rule. If the child was not valid, the heuristics would then return **false** without even invoking the replacement rule, under the assumption that the parent was valid to start with. If the child was valid, the replacement rule would be invoked as usual.

```
boolean DETERMINISTICREPLACEMENT(f(p), f(c), g_C)
begin
    if f(c) > f(p) then return true
    else if f(c) = f(p) return FLIPCOIN(1/2)
    else return false
end

boolean PROBABILISTICREPLACEMENT(f(p), f(c), g_C)
begin
    p ← f(c) / (f(c)+f(p))
    return FLIPCOIN(p)
end

boolean BOLTZMANNREPLACEMENT(f(p), f(c), g_C)
begin
    T_C ← T_0 × exp(c_C × g_C) {c_C is a constant; T_0 initial temperature}
    pScore ← exp((f(p) − c_S) / T_C) {c_S is a constant}
    cScore ← exp((f(c) − c_S) / T_C)
    p ← cScore / (pScore + cScore)
    return FLIPCOIN(p)
end

boolean METROPOLISREPLACEMENT(f(p), f(c), g_C)
begin
    △f ← f(c) - f(p)
    if △f ≥ 0 then return true
    else { △f < 0 }
        r ← RANDOMDOUBLE(0,1)
        T_C ← T_0 × exp(c_C × g_C)
        if r < exp(△f / T_C) then return true
        else return false
    end
end

boolean NOISYREPLACEMENT(f(p), f(c), g_C)
begin
   return FLIPCOIN(1/2)
end
```

Figure 4: Different replacement rules used in the crowding GA. Each rule has as input the parent's fitness $f(\boldsymbol{p})$, the child's fitness $f(\boldsymbol{c})$, and the generation counter g_C. Each rule returns **true** if the child's fitness $f(\boldsymbol{c})$ is better than the parent's fitness $f(\boldsymbol{p})$, according to the replacement rule, else the rule returns **false**.

4.3 Replacement Rules

A replacement rule R determines how a crowding GA picks the winner in a competition between two individuals. Such rules are used both in SIMPLESTEP and in CROWDINGSTEP. Without loss of generality, we denote the individuals input to R a parent $\boldsymbol{p}$, with fitness $f(\boldsymbol{p})$, and a child $\boldsymbol{c}$, with fitness $f(\boldsymbol{c})$. If R returns **true** then $\boldsymbol{c}$ is the winner (giving $\boldsymbol{w} \leftarrow \boldsymbol{c}$ in CROWDINGSTEP), else $\boldsymbol{p}$ is the winner (giving $\boldsymbol{w} \leftarrow \boldsymbol{p}$). Example replacement rules are presented in Figure 4. In these rules, FLIPCOIN(p) simulates a binomial random variable with parameter p while RANDOMDOUBLE(a, b) simulates a uniform random variable with parameters a and b and produces an output r such that $a \leq r \leq b$. The probabilistic crowding approach is based on deterministic crowding (Mahfoud, 1995); in this section we focus on the DETERMINISTICREPLACEMENT rule of deterministic crowding

and the PROBABILISTICREPLACEMENT rule of probabilistic crowding. We also present more briefly other replacement rules, in particular BOLTZMANNREPLACEMENT, METROPOLIS REPLACEMENT, and NOISYREPLACEMENT.

It also turns out that different replacement rules can be combined; we return to this in Section 6.

4.3.1 Deterministic Crowding Replacement Rule

DETERMINISTICREPLACEMENT (abbreviated R_{D}) implements the deterministic crowding replacement rule (Mahfoud, 1995) in our framework. This replacement rule always picks the individual with the higher fitness score, be it $f(\boldsymbol{c})$ (fitness of the child $\boldsymbol{c}$) or $f(\boldsymbol{p})$ (fitness of the parent $\boldsymbol{p}$). The DETERMINISTICREPLACEMENT rule gives the following probability for $\boldsymbol{c}$ winning the tournament:

$$p_{\boldsymbol{c}} = p(\boldsymbol{c}) = \begin{cases} 1 & \text{if } f(\boldsymbol{c}) > f(\boldsymbol{p}) \\ \frac{1}{2} & \text{if } f(\boldsymbol{c}) = f(\boldsymbol{p}) \\ 0 & \text{if } f(\boldsymbol{c}) < f(\boldsymbol{p}) \end{cases}. \tag{7}$$

4.3.2 Probabilistic Crowding Replacement Rule

PROBABILISTICREPLACEMENT (R_{P}) implements the probabilistic crowding approach (Mengshoel and Goldberg, 1999) in our framework. Let $\boldsymbol{c}$ and $\boldsymbol{p}$ be the two individuals that have been matched to compete. In probabilistic crowding, $\boldsymbol{c}$ and $\boldsymbol{p}$ compete in a probabilistic tournament. The probability of $\boldsymbol{c}$ winning is given by:

$$p_{\boldsymbol{c}} = p(\boldsymbol{c}) = \frac{f(\boldsymbol{c})}{f(\boldsymbol{c}) + f(\boldsymbol{p})}, \tag{8}$$

where f is the fitness function.

After the probabilistic replacement rule was first introduced (Mengshoel and Goldberg, 1999), a continuous variant of probabilistic crowding has successfully been developed and applied to hard multimodal optimization problems in high-dimensional spaces (Ballester and Carter, 2003, 2004, 2006).

4.3.3 Other Replacement Rules

In addition to those already discussed, the following replacement rules have been identified. Note that some of these latter rules refer to global variables—specifically initial temperature T_0, cooling constant c_{C}, and scaling constant c_{S}—whose values are application-specific and assumed to be set appropriately.

- BOLTZMANNREPLACEMENT (abbreviated R_{B}) picks the child $\boldsymbol{c}$ proportionally to its score cScore, and the parent $\boldsymbol{p}$ proportionally to its score pScore. The constant c_S is a scaling factor that prevents the exponent from getting too large. A good default value is $c_{\mathrm{S}} = 0$. T_{C} is the temperature, which decreases as the generation g_{C} of the GA increases. Boltzmann replacement has also been used in PRSA (Mahfoud and Goldberg, 1995).

- METROPOLISREPLACEMENT (R_{M}) always picks the child $\boldsymbol{c}$ if there is a non-decrease in Δf, else it will hold a probabilistic tournament where either child $\boldsymbol{c}$ or parent $\boldsymbol{p}$ wins. This rule was introduced in 1953 in what is now known as the Metropolis

algorithm, an early Monte Carlo approach (Metropolis et al., 1953) which was later generalized (Hastings, 1970).

- NoisyReplacement (R_R) is a pure noise replacement rule, similar to approaches found to be powerful as a complement to other replacement rules in local search for finding a satisfying assignment in propositional logic (Selman et al., 1994) and in computing most probable explanations in Bayesian networks (Mengshoel, 1999).

There is an important difference between, on the one hand, applications of replacement rules in statistical physics and, on the other hand, applications of replacement rules in optimization using evolutionary algorithms. In statistical physics, there is a need to obtain the Boltzmann distribution at equilibrium in order to properly model physical reality. Since both the BoltzmannReplacement and MetropolisReplacement rules have this property, they are used for Monte Carlo simulation in statistical physics (Metropolis et al., 1953; Newman and Barkema, 1999). In optimization we are not, however, necessarily restricted to the Boltzmann distribution at equilibrium. We are therefore free to investigate replacement rules other than BoltzmannReplacement and MetropolisReplacement, as we indeed do in this article.

By combining different steps and replacement rules we obtain different crowding GAs as follows: SimpleStep with ProbabilisticReplacement gives the SimplePCGA; CrowdingStep with ProbabilisticReplacement gives the GeneralPCGA; SimpleStep with DeterministicReplacement gives the SimpleDCGA; and CrowdingStep with DeterministicReplacement gives the GeneralDCGA.

5 Analysis of Crowding

Complementing the presentation of algorithms in Section 4, we now turn to analysis. One of the most important questions to ask about a niching algorithm is what the characteristics of its steady-state (equilibrium) distribution are. In particular, we are interested in this for niches. The notation $\boldsymbol{x} \in \mathbb{X}$ will below be used to indicate that individual $\boldsymbol{x}$ is a member of niche $\mathbb{X} \subseteq \{0,1\}^m$.

Definition 4 (Niching rule): *Let q be the number of niches, let $\mathbb{X}_i$ be the i-th niche, and let $\boldsymbol{x}_i \in \mathbb{X}_i$. Further, let f_i be a measure of fitness of individuals in niche $\mathbb{X}_i$ (typically fitness of the best fit individual or average fitness of all individuals). The* niching rule *is defined as*

$$\eta_i = \frac{f_i}{\sum_{j=1}^{q} f_j} = \frac{f(\boldsymbol{x}_i)}{\sum_{j=1}^{q} f(\boldsymbol{x}_j)}. \tag{9}$$

We note that the niching rule gives proportions $0 \leq \eta_i \leq 1$. Analytically, it gives an allocation of $\eta_i \times n$ individuals to niche $\mathbb{X}_i$, where n is the population size. The rule can be derived from the sharing rule (Goldberg and Richardson, 1987), and is considered an idealized baseline for niching algorithms. In the following, we will see that the behavior of probabilistic crowding is related to the niching rule.

In the rest of this section, we first provide an analysis of our crowding approach. Two types of analysis are provided: at steady state and of the form of convergence of the population. We assume some variation operator, typically mutation or crossover. In the analysis we assume one representative per niche; for example, if the niche is $\mathbb{X}$, the representative is $\boldsymbol{x} \in \mathbb{X}$. Using difference equations, we perform a deterministic

analysis, thus approximating the stochastic sampling in a crowding GA. Applying the theoretical framework, the replacements rules of probabilistic crowding (PROBABILISTICREPLACEMENT) and deterministic crowding (DETERMINISTREPLACEMENT) are studied in more detail. Both the special case with two niches as well as the more general case with several niches are analyzed. The third area we discuss in this section is population sizing.

5.1 Two Niches, Same Jump Probabilities

We first discuss the case of two niches $\mathbb{X}$ and $\mathbb{Y}$. This is admittedly a restriction of the setting with an arbitrary number of niches, but a separate investigation is warranted for several reasons. First, some fitness functions may have exactly two optimal (maxima or minima) points, or one optimal point and another almost-optimal point, and one may want to find both of them. Second, one may use the two-niche case as an abstraction of the multi-niche case, where one niche (say $\mathbb{X}$) is an actual niche while the second niche (say $\mathbb{Y}$) is used to lump together all other niches. Third, the two-niche case is a stepping stone for the analysis of more than two niches; such further analysis follows below.

In the two-niche case, suppose we have a variation operator that results in two types of jumps: short jumps and long jumps. When an individual is treated with a short jump by the GA it stays within its niche, when it is treated with a long jump it moves to the other niche. The probabilities for undertaking short and long jumps are p_s and p_ℓ respectively, where $p_s + p_\ell = 1$. That is, we either jump short or long. Generally, we assume non-degenerate probabilities $0 < p_s, p_\ell < 1$ in this article.

Consider parent $\boldsymbol{p}$ and child $\boldsymbol{c}$ constructed by either SIMPLESTEP or CROWDINGSTEP. Further, consider how $\mathbb{X}$ can gain or maintain individuals from one generation to the next. Let $\boldsymbol{w}$ be the winner of the local tournament, and suppose that we focus on $\boldsymbol{w} \in \mathbb{X}$. By the law of total probability, we have the following probability of the winner $\boldsymbol{w}$, either parent $\boldsymbol{p}$ or child $\boldsymbol{c}$, ending up in the niche $\mathbb{X}$:

$$\Pr(\boldsymbol{w} \in \mathbb{X}) = \sum_{\mathbb{A},\mathbb{B}\in\{\mathbb{X},\mathbb{Y}\}} \Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{A}, \boldsymbol{c} \in \mathbb{B}). \tag{10}$$

There are clearly four combinations possible for $\boldsymbol{p} \in \mathbb{A}$, $\boldsymbol{c} \in \mathbb{B}$ in Equation (10). By using Bayes' rule for each combination, we obtain the following:

$$\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{X}) = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{X}) \Pr(\boldsymbol{c} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{X}) \tag{11}$$

$$\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) \Pr(\boldsymbol{c} \in \mathbb{Y}, \boldsymbol{p} \in \mathbb{X}) \tag{12}$$

$$\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X}) = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X}) \Pr(\boldsymbol{c} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{Y}) \tag{13}$$

$$\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{Y}) = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{Y}) \Pr(\boldsymbol{c} \in \mathbb{Y}, \boldsymbol{p} \in \mathbb{Y}). \tag{14}$$

Here, Equation (11) represents a short jump inside $\mathbb{X}$; Equation (12) represents a long jump from $\mathbb{X}$ to $\mathbb{Y}$; Equation (13) represents a long jump from $\mathbb{X}$ to $\mathbb{Y}$; and Equation (14) represents a short jump inside $\mathbb{Y}$.

Before continuing our analysis, we introduce the following assumptions and definitions:

$$p_s = \Pr(\boldsymbol{c} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}) = \Pr(\boldsymbol{c} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{Y})$$
$$p_\ell = \Pr(\boldsymbol{c} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}) = \Pr(\boldsymbol{c} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{X})$$

$$p_x = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X})$$
$$p_y = \Pr(\boldsymbol{w} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) = \Pr(\boldsymbol{w} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X}).$$

In words, p_s is the probability of a short jump (either inside $\mathbb{X}$ or $\mathbb{Y}$); p_ℓ is the probability of a long jump (either from $\mathbb{X}$ to $\mathbb{Y}$ or in the opposite direction); and p_x is the probability of $\boldsymbol{w} \in \mathbb{X}$ given that the parents are in different niches. That is, p_x is the probability of an individual in $\mathbb{X}$ winning the local tournament. Similarly, p_y is the probability of an individual in $\mathbb{Y}$ winning.

Obviously, Equation (14) is zero and will not be considered further below. Excluding Equation (14) there are three cases, which we now consider in turn. The first case of Equation (11) involves $\boldsymbol{p} \in \mathbb{X}$. Specifically, a short jump is made and the child $\boldsymbol{c}$ stays in the parent $\boldsymbol{p}$'s niche $\mathbb{X}$. With respect to $\mathbb{X}$, it does not matter whether $\boldsymbol{p}$ or $\boldsymbol{c}$ wins since both are in the same niche, and by using Bayes' rule we get for Equation (11):

$$\begin{aligned}\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{X}) &= \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{X}) \Pr(\boldsymbol{c} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}) \Pr(\boldsymbol{p} \in \mathbb{X}) \\ &= p_s \Pr(\boldsymbol{p} \in \mathbb{X}). \end{aligned} \tag{15}$$

The second case, of Equation (12), is that the child jumps long from $\mathbb{X}$ to $\mathbb{Y}$ and loses, and we get:

$$\begin{aligned}\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) &= \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}) \Pr(\boldsymbol{c} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{X}) \Pr(\boldsymbol{p} \in \mathbb{X}) \\ &= p_x p_\ell \Pr(\boldsymbol{p} \in \mathbb{X}). \end{aligned} \tag{16}$$

The third and final case, of Equation (13), involves $\boldsymbol{p} \in \mathbb{Y}$. Now, gain for niche $\mathbb{X}$ takes place when the child $\boldsymbol{c}$ jumps to $\mathbb{X}$ and wins over $\boldsymbol{p}$. Formally, we obtain:

$$\begin{aligned}\Pr(\boldsymbol{w} \in \mathbb{X}, \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X}) &= \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}, \boldsymbol{c} \in \mathbb{X}) \Pr(\boldsymbol{c} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{Y}) \Pr(\boldsymbol{p} \in \mathbb{Y}) \\ &= p_x p_\ell \Pr(\boldsymbol{p} \in \mathbb{Y}). \end{aligned} \tag{17}$$

By substituting Equations (15), (16), and (17) into Equation (10) we get the following:

$$\begin{aligned}\Pr(\boldsymbol{w} \in \mathbb{X}) &= p_s \Pr(\boldsymbol{p} \in \mathbb{X}) + p_x p_\ell \Pr(\boldsymbol{p} \in \mathbb{X}) + p_x p_\ell \Pr(\boldsymbol{p} \in \mathbb{Y}) \\ &= \Pr(\boldsymbol{p} \in \mathbb{X}) - p_\ell \Pr(\boldsymbol{p} \in \mathbb{X}) + p_\ell p_x. \end{aligned} \tag{18}$$

We will solve this equation in two ways, namely by considering the steady state (or equilibrium) and by obtaining a closed form formula. Assuming that a steady state exists, we have

$$\Pr(\boldsymbol{p} \in \mathbb{X}) = \Pr(\boldsymbol{w} \in \mathbb{X}). \tag{19}$$

Substituting Equation (19) into Equation (18) gives

$$\Pr(\boldsymbol{w} \in \mathbb{X}) = \Pr(\boldsymbol{w} \in \mathbb{X}) - p_\ell \Pr(\boldsymbol{w} \in \mathbb{X}) + p_\ell p_x \tag{20}$$

which can be simplified to

$$\Pr(\boldsymbol{w} \in \mathbb{X}) = p_x \tag{21}$$

where p_x depends on the replacement rule being used as follows.

For PROBABILISTICREPLACEMENT, we use Equation (8) to obtain for Equation (21)

$$\Pr(\boldsymbol{w} \in \mathbb{X}) = \frac{f(\boldsymbol{x})}{f(\boldsymbol{x}) + f(\boldsymbol{y})} \tag{22}$$

where $\boldsymbol{x} \in \mathbb{X}$, $\boldsymbol{y} \in \mathbb{Y}$. In other words, we get the niching rule of Equation (9) at steady state.

Using DETERMINISTICREPLACEMENT, suppose $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$. We obtain for Equation (21)

$$\begin{aligned} \Pr(\boldsymbol{w} \in \mathbb{X}) &= 1 \quad \text{if } f(\boldsymbol{x}) > f(\boldsymbol{y}) \\ \Pr(\boldsymbol{w} \in \mathbb{X}) &= \frac{1}{2} \quad \text{if } f(\boldsymbol{x}) = f(\boldsymbol{y}). \end{aligned}$$

We now turn to obtaining a closed form formula. By assumption we have two niches, $\mathbb{X}$ and $\mathbb{Y}$, and the proportions of individuals of interest at time t are denoted $X(t)$ and $Y(t)$ respectively.[5] Note that $X(t) + Y(t) = 1$ for any t. Now, $\boldsymbol{w} \in \mathbb{X}$ is equivalent to $X(t+1) = 1$, where $X(t+1)$ is an indicator random variable for niche $\mathbb{X}$ for generation $t+1$, and we note that

$$\Pr(\boldsymbol{w} \in \mathbb{X}) = \Pr(X(t+1) = 1) = E(X(t+1)). \tag{23}$$

The last equality holds because the expectation of $X(t+1)$ is $E(X(t+1)) = \sum_{i=0}^{1} i \Pr(X(t+1) = i) = \Pr(X(t+1) = 1)$. Along similar lines, $\Pr(\boldsymbol{p} \in \mathbb{X}) = \Pr(X(t) = 1) = E(X(t))$.

Considering two expected niche proportions $E(X(t))$ and $E(Y(t))$, we have these two difference equations:

$$\begin{aligned} E(X(t+1)) &= p_{\mathrm{s}} E(X(t)) + p_{\ell} p_{\boldsymbol{x}} E(X(t)) + p_{\ell} p_{\boldsymbol{x}} E(Y(t)) \\ E(Y(t+1)) &= p_{\mathrm{s}} E(Y(t)) + p_{\ell} p_{\boldsymbol{y}} E(Y(t)) + p_{\ell} p_{\boldsymbol{y}} E(X(t)). \end{aligned} \tag{24}$$

The solution to the above system of difference equations can be written as:

$$E(X(t)) = p_{\boldsymbol{x}} + p_{\mathrm{s}}^t E(X(0)) - p_{\mathrm{s}}^t p_{\boldsymbol{x}} E(X(0)) - p_{\mathrm{s}}^t p_{\boldsymbol{x}} E(Y(0)) \tag{25}$$

$$E(Y(t)) = p_{\boldsymbol{y}} - p_{\mathrm{s}}^t E(X(0)) + p_{\mathrm{s}}^t p_{\boldsymbol{x}} E(X(0)) + p_{\mathrm{s}}^t p_{\boldsymbol{x}} E(Y(0)) \tag{26}$$

where $t = 0$ is the time of the initial generation.

For PROBABILISTICREPLACEMENT we see how, as $t \to \infty$ and assuming $p_{\mathrm{s}} < 1$, we get the niching rule, Equation (9), expressed as $p_{\boldsymbol{x}}$ and $p_{\boldsymbol{y}}$, for both niches. More formally, $\lim_{t\to\infty} E(X(t)) = p_{\boldsymbol{x}}$ and $\lim_{t\to\infty} E(Y(t)) = p_{\boldsymbol{y}}$. In other words, initialization does not affect the fact that the niching rule is achieved in the limit when PROBABILISTICREPLACEMENT is used for crowding.

We now turn to the effect of the initial population, which is important before equilibrium is reached. Above, $E(X(0))$ and $E(Y(0))$ reflect the initialization algorithm used. Assuming initialization uniformly at random, we let in the initial population

[5] Instead of using the proportion of a population allocated to a niche, one can base the analysis on the number of individuals in a niche. The analysis is quite similar in the two cases, and in the analysis in this article we have generally used the former proportional approach.

$E(X(0)) = E(X(Y(0)) = \frac{1}{2}$. This gives the following solutions for Equations (25) and (26):

$$E(X(t)) = p_{\boldsymbol{x}} + \left(\frac{1}{2} - p_{\boldsymbol{x}}\right) p_{\mathrm{s}}^t \tag{27}$$

$$E(Y(t)) = p_{\boldsymbol{y}} + \left(\frac{1}{2} - p_{\boldsymbol{y}}\right) p_{\mathrm{s}}^t. \tag{28}$$

Again, under the $p_{\mathrm{s}} < 1$ assumption already mentioned, we see how $p_{\boldsymbol{x}}$ and $p_{\boldsymbol{y}}$ result as $t \to \infty$. Also note in Equations (27) and (28) that a smaller p_{s}, and consequently a larger $p_{\ell} = 1 - p_{\mathrm{s}}$, gives faster convergence to the niching rule at equilibrium.

We now turn to DeterministicReplacement. Suppose that $p_{\boldsymbol{x}} = 0$ and $p_{\boldsymbol{y}} = 1$, for example, we may have $f(\boldsymbol{x}) = 1$ and $f(\boldsymbol{y}) = 4$. Substituting into Equation (27) gives

$$E(X(t)) = \frac{1}{2} p_s^t \tag{29}$$

$$E(Y(t)) = 1 - \frac{1}{2} p_s^t \tag{30}$$

which provides a (to our knowledge) novel result regarding the analysis of convergence for deterministic crowding, thus improving the understanding of how this algorithm operates. Under the assumption $p_{\mathrm{s}} < 1$ we get $\lim_{t\to\infty} E(X(t)) = 0$ and $\lim_{t\to\infty} E(Y(t)) = 1$ for Equations (29) and (30), respectively. In this example, and except for the degenerate case $p_{\boldsymbol{x}} = 0$ and $p_{\boldsymbol{y}} = 1$, deterministic crowding gives a much stronger selection pressure than probabilistic crowding. Using DeterministicReplacement, a more fit niche (here $\mathbb{Y}$) will in the limit $t \to \infty$ win over a less fit niche (here $\mathbb{X}$). Using ProbabilisticReplacement, on the other hand, both niches are maintained—subject to noise—in the limit $t \to \infty$. Considering the operation of DeterministicReplacement, the main difference with ProbabilisticReplacement is that there is no restorative pressure—thus niches may get lost under DeterministicReplacement even though they have substantial fitness.

5.2 Two Niches, Different Jump Probabilities

Here we relax the assumption of equal jump probabilities for the two niches $\mathbb{X}$ and $\mathbb{Y}$. Rather than jump probabilities p_{s} and p_{ℓ}, we have jump probabilities p_{ij} for jumping from niche $\mathbb{X}_i$ to niche $\mathbb{X}_j$, where $i, j \in \{0, 1\}$. We use the notation $E(X_i(t))$ for the expected proportion of individuals in niche $\mathbb{X}_i$ at time t, and let p_i be the probability of the i-th niche winning in a local tournament. The facts $p_{11} + p_{12} = 1$ and $p_{21} + p_{22} = 1$ are used below, too.

We obtain the following expression for $E(X_1(t+1))$, using reasoning similar to that used for Equation (18):

$$\begin{aligned} E(X_1(t+1)) &= p_{11}E(X_1(t)) + p_{12}p_1E(X_1(t)) + p_{21}p_1E(X_2(t)) \\ &= p_{11}E(X_1(t)) + (1 - p_{11})p_1E(X_1(t)) + p_{21}p_1(1 - E(X_1(t))) \\ &= p_{11}E(X_1(t)) + p_1E(X_1(t)) - p_{11}p_1E(X_1(t)) - p_{21}p_1E(X_1(t)) + p_{21}p_1. \end{aligned} \tag{31}$$

At steady state we have $E(X_1(t+1)) = E(X_1(t)) = E(X_1)$, leading to

$$E(X_1) = p_{11}E(X_1) + p_1E(X_1) - p_{11}p_1E(X_1) - p_{21}p_1E(X_1) + p_{21}p_1$$

which after some manipulation simplifies to the following allocation ratio for niche $\mathbb{X}_1$

$$E(X_1) = \frac{p_1}{p_1 + \frac{p_{12}}{p_{21}}p_2} = \frac{p_1}{p_1 + \xi_{12}p_2}. \tag{32}$$

Here, $\xi_{12} := \frac{p_{12}}{p_{21}}$ is denoted the transmission ratio from $\mathbb{X}_1$ to $\mathbb{X}_2$. In general, we say that ξ_{ij} is the transmission ratio from niche $\mathbb{X}_i$ to $\mathbb{X}_j$. Clearly, ξ_{12} is large if the transmission of individuals from $\mathbb{X}_1$ into $\mathbb{X}_2$ is large relative to the transmission from $\mathbb{X}_2$ into $\mathbb{X}_1$.

Let $\boldsymbol{x}_1 \in \mathbb{X}_1$ and $\boldsymbol{x}_2 \in \mathbb{X}_2$. Assuming ProbabilisticReplacement and using Equation (8) we obtain $p_1 = \frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_1)+f(\boldsymbol{x}_2)}$ and $p_2 = \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_1)+f(\boldsymbol{x}_2)}$. Substituting these values for p_1 and p_2 into Equation (32) and simplifying gives

$$E(X_1) = \frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_1) + \xi_{12}f(\boldsymbol{x}_2)}. \tag{33}$$

For two niches, Equation (33) is clearly a generalization of the niching rule in Equation (9); just put $\xi_{12} = 1$ in Equation (33) to obtain Equation (9).

The size of a niche as well as the operators used may have an impact on p_{12} and p_{21} and thereby also on ξ_{12} and ξ_{21}. Comparing Equations (9) and (33), we note how $\xi_{12} > 1$ means that niche $\mathbb{X}_2$ will have a larger subpopulation at equilibrium than under the niching rule, giving $\mathbb{X}_1$ a smaller subpopulation, while $\xi_{12} < 1$ means that $\mathbb{X}_2$'s subpopulation at equilibrium will be smaller than under the niching rule, giving $\mathbb{X}_1$ a larger subpopulation.

Along similar lines, the ratio for niche $\mathbb{X}_2$ turns out to be

$$E(X_2) = \frac{p_2}{\frac{p_{21}}{p_{12}}p_1 + p_2} = \frac{p_2}{\xi_{21}p_1 + p_2} \tag{34}$$

with $\xi_{21} := \frac{p_{21}}{p_{12}}$.

Note that values for ξ_{12} and ξ_{21}, or more generally ξ_{ij} for the transmission ratio from niche i to j, will be unknown. So one cannot use known values for ξ_{12} and ξ_{21} in the equations above. However, it is possible to estimate transmission ratios using sampling or one may use worst-case or average-case values.

Finally, we note that the same result as in Equations (32) and (34) can be established by solving these two simultaneous difference equations:

$$E(X_1(t+1)) = p_{11}E(X_1(t)) + p_{12}p_1E(X_1(t)) + p_{21}p_1E(X_2(t))$$
$$E(X_2(t+1)) = p_{22}E(X_2(t)) + p_{21}p_2E(X_2(t)) + p_{12}p_2E(X_1(t))$$

which yields fairly complex solutions which can be solved by eliminating all terms with generation t in the exponent. These solutions can then be simplified, giving exactly the same result as above.

5.3 Multiple Niches, Different Jump Probabilities

We now generalize from two to $q \geq 2$ niches. Let the probability of transfer from the i-th to j-th niche by the crowding algorithm be p_{ij}, where $\sum_{j=1}^{q} p_{ij} = 1$. Let the probability of an individual $\boldsymbol{x}_i \in \mathbb{X}_i$ occurring at time t be $p_i(t)$, and let its probability of winning over an individual $\boldsymbol{x}_j \in \mathbb{X}_j$ in a local tournament be p^*_{ij}. The expression for p^*_{ij} depends on the replacement rule as we will discuss later in this section. We can now set up a system of q difference equations of the following form by letting $i \in \{1, \ldots, q\}$:

$$p_i(t+1) = \sum_{j \neq i} p_{ij} p^*_{ij} p_i(t) + \sum_{j \neq i} p_{ji} p^*_{ij} p_j(t) + p_{ii} p_i(t). \tag{35}$$

In words, $p_{ij} p^*_{ij} p_i(t)$ represents transmission of individuals from $\mathbb{X}_i$ to $\mathbb{X}_j$ and $p_{ji} p^*_{ij} p_j(t)$ represents transmission of individuals from $\mathbb{X}_j$ to $\mathbb{X}_i$. Unfortunately, these equations are hard to solve. But by introducing the assumption of local balance (known as detailed balance in physics; Laarhoven and Aarts, 1987), progress can be made. The condition is (Neal, 1993, p. 37):

$$p_i p_{ij} p^*_{ji} = p_j p_{ji} p^*_{ij}. \tag{36}$$

The local balance assumption is that individuals (or states, or niches) are in balance: The probability of an individual $\boldsymbol{x}_i$ being transformed into another individual $\boldsymbol{x}_j$ is the same as the probability of the second individual $\boldsymbol{x}_j$ being transformed into the first individual $\boldsymbol{x}_i$. We can assume this is for a niche rather than for an individual, similar to what we did above, thus giving Equation (36). On the left-hand side of Equation (36) we have the probability of escaping niche $\mathbb{X}_i$, on the right-hand side of Equation (36) we have the probability of escaping niche $\mathbb{X}_j$. Simple rearrangement of Equation (36) gives

$$p_i = \frac{p_{ji} p^*_{ij}}{p_{ij} p^*_{ji}} p_j = \xi_{ji} \frac{p^*_{ij}}{p^*_{ji}} p_j \tag{37}$$

where $\xi_{ji} := \frac{p_{ji}}{p_{ij}}$. Here, $\frac{p^*_{ij}}{p^*_{ji}}$ depends on the replacement rule used.

Using the framework introduced above, we analyze the ProbabilisticReplacement rule presented in Section 4 and in Figure 4. Specifically, for two niches $\mathbb{X}_i$ and $\mathbb{X}_j$ we have for p^*_{ij} in Equation (37)

$$p^*_{ij} = \frac{f(\boldsymbol{x}_i)}{f(\boldsymbol{x}_i) + f(\boldsymbol{x}_j)} \tag{38}$$

where $\boldsymbol{x}_i \in \mathbb{X}_i$ and $\boldsymbol{x}_j \in \mathbb{X}_j$. Using Equation (38) and a similar expression for p^*_{ji}, we substitute for p^*_{ij} and p^*_{ji} in Equation (37) and obtain

$$p_i = \frac{p_{ji} p^*_{ij}}{p_{ij} p^*_{ji}} p_j = \xi_{ji} \frac{f(\boldsymbol{x}_i)}{f(\boldsymbol{x}_j)} p_j. \tag{39}$$

We now consider the k-th niche, and express all other niches, using Equation (39), in terms of this niche. In particular, we express an arbitrary proportion p_i using a particular

proportion p_k:

$$p_i = \xi_{ki} \frac{f(\boldsymbol{x}_i)}{f(\boldsymbol{x}_k)} p_k. \tag{40}$$

Now, we introduce the fact that $\sum_{i=1}^{q} p_i = 1$, where q is the number of niches:

$$\xi_{k1} \frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_k)} p_k + \xi_{k2} \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_k)} p_k + \cdots + p_k + \cdots + \xi_{kq} \frac{f(\boldsymbol{x}_q)}{f(\boldsymbol{x}_k)} p_k = 1. \tag{41}$$

Solving for p_k in Equation (41) gives

$$p_k = \frac{1}{\xi_{k1} \frac{f(\boldsymbol{x}_1)}{f(\boldsymbol{x}_k)} + \xi_{k2} \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_k)} + \cdots + 1 + \cdots + \xi_{kq} \frac{f(\boldsymbol{x}_q)}{f(\boldsymbol{x}_k)}} \tag{42}$$

and we next use the fact that

$$\frac{f(\boldsymbol{x}_k)}{f(\boldsymbol{x}_k)} \xi_{kk} = 1 \tag{43}$$

where we set $\xi_{kk} := 1$. Substituting Equation (43) into Equation (42) and simplifying gives

$$p_k = \frac{f(\boldsymbol{x}_k)}{\sum_{i=1}^{q} \xi_{ki} f(\boldsymbol{x}_i)}. \tag{44}$$

Notice how the transmission ratio ξ_{ki} from $\mathbb{X}_k$ to $\mathbb{X}_i$ generalizes the transmission ratio ξ_{12} from Equation (32). Equation (44) is among the most general theoretical result on probabilistic crowding presented in this article; it generalizes the niching rule of Equation (9) (see also Mahfoud, 1995, p. 157). The niching rule applies to sharing with roulette-wheel selection (Mahfoud, 1995), and much of that approach to analyzing niching GAs can now be carried over to probabilistic crowding.

5.4 Noise and Population Sizing

Suppose that the population is in equilibrium. Each time an individual is sampled from or placed into the population, it can be considered a Bernoulli trial. Specifically, suppose it is a Bernoulli trial with a certain probability p of the winner $\boldsymbol{w}$ being taken from or placed into a particular niche $\mathbb{X}$. We now form for each trial an indicator random variable S_i as follows:

$$S_i = \begin{cases} 0 & \text{if } \boldsymbol{w} \notin \mathbb{X} \\ 1 & \text{if } \boldsymbol{w} \in \mathbb{X} \end{cases}.$$

Taking into account all n individuals in the population, we form the random variable $B = \sum_{i=1}^{n} S_i$; clearly B has a binomial probability density BinomialDen$(x; n, p)$. The above argument can be extended to an arbitrary number of niches. Again, consider the crowding GA's operation as n Bernoulli trials. Now, the probability of picking from the k-th niche $\mathbb{X}_k$ is given by p_k in Equation (44). This again gives a binomial distribution BinomialDen$(x; n, p_k)$, where $\mu_k = np_k$ and $\sigma_k^2 = np_k(1 - p_k)$.

We now turn to our novel population sizing result. To derive population sizing results for crowding, we consider again the population at equilibrium. Then, at equilibrium, we require that with high probability, we shall find individuals from the desired niches in the population. We consider the joint distribution over the number of members in all q niches possible in the population, $\Pr(\boldsymbol{B}) = \Pr(B_1, \ldots, B_q)$. In general, we are interested in the joint probability that each of the q niches has at least a certain (niche-dependent) number of members b_i, for $1 \leq i \leq q$: $\Pr(B_1 \geq b_1, \ldots\ldots, B_q \geq b_q)$. An obvious population size lower bound is then $n \geq \sum_{i=1}^{q} b_i$. Of particular interest are the κ fittest niches $\mathbb{X}_1, \ldots, \mathbb{X}_\kappa$ among all niches $\mathbb{X}_1, \ldots, \mathbb{X}_q$, and without loss of generality we assume an ordering in which the κ fittest niches come first. Specifically, we are interested in the probability that each of the κ niches has a positive member count, giving

$$\Pr\left(B_1 > 0, \ldots, B_\kappa > 0, B_{\kappa+1} \geq 0, \ldots, B_q \geq 0\right) = \Pr\left(B_1 > 0, \ldots, B_\kappa > 0\right) \tag{45}$$

since a random variable B_i representing the i-th niche is clearly non-negative. Assuming independence for simplicity, we obtain for Equation (45)

$$\Pr\left(B_1 > 0, \ldots, B_\kappa > 0\right) = \prod_{i=1}^{\kappa} \Pr\left(B_i > 0\right) = \prod_{i=1}^{\kappa} \left(1 - \Pr\left(B_i = 0\right)\right). \tag{46}$$

Further progress can be made by assuming that B_i follows a binomial distribution with probability p_i as discussed above. For binomial B we have $\Pr(B = j) = \binom{n}{j} p^j (1-p)^{n-j}$. Putting $j = 0$ in this expression for $\Pr(B = j)$ gives for Equation (46):

$$\prod_{i=1}^{\kappa} \left(1 - \Pr\left(B_i = 0\right)\right) = \prod_{i=1}^{\kappa} \left(1 - (1 - p_i)^n\right). \tag{47}$$

Simplifying further, a conservative lower bound can be derived by focusing on the least fit niche among the κ desired niches. Without loss of generality, assume that $p_\kappa \leq p_{\kappa-1} \leq \cdots \leq p_1$. Consequently, $(1 - p_\kappa) \geq (1 - p_{\kappa-1}) \geq \cdots \geq (1 - p_1)$ and therefore since $n \geq 1$

$$(1 - (1 - p_\kappa)^n) \leq (1 - (1 - p_{\kappa-1})^n) \leq \cdots \leq (1 - (1 - p_1)^n)$$

from which it is easy to see that

$$\Pr\left(B_1 > 0, \ldots, B_\kappa > 0\right) = \prod_{i=1}^{\kappa} \left(1 - (1 - p_i)^n\right) \geq \left(1 - (1 - p_\kappa)^n\right)^\kappa. \tag{48}$$

In other words, we have the following lower bound on the joint probability $\gamma(n, \kappa, p_\kappa)$:

$$\gamma(n, \kappa, p_\kappa) := \Pr\left(B_1 > 0, \ldots, B_\kappa > 0\right) \geq \left(1 - (1 - p_\kappa)^n\right)^\kappa. \tag{49}$$

For simplicity, we often say just γ instead of $\gamma(n, \kappa, p_\kappa)$. This is an important result since it ties together positive niche counts in the κ most fit niches, smallest niche probability p_κ, and population size n.

Solving for n in Equation (49) gives the following population sizing result.

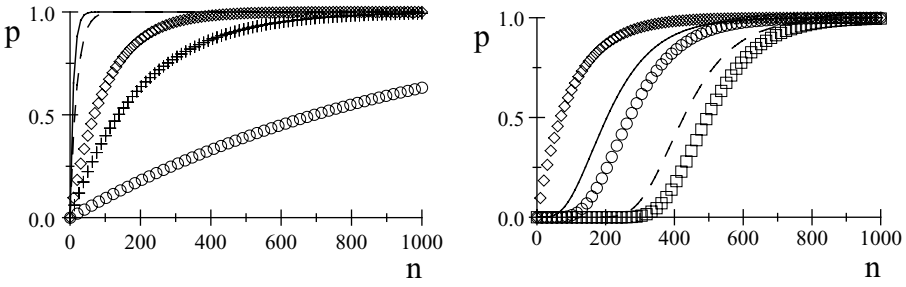

Figure 5: The effect of varying the population size n (along the x-axis), the desired number of niches κ, and the smallest niche probability p_κ on the lower bound $p = \gamma(n, \kappa, p_\kappa)$ (along the y-axis) for the joint niche count probability. *Left:* Here we keep constant $\kappa = 1$ and vary the population size n as well as the niche probability p_κ at steady state: $p_\kappa = 0.1$ (solid line), $p_\kappa = 0.05$ (dashed line), $p_\kappa = 0.01$ (diamond line), $p_\kappa = 0.005$ (cross line), and $p_\kappa = 0.001$ (circle line). *Right:* Here we keep constant $p_\kappa = 0.01$ and vary the population size n as well as the number of maintained niches κ: $\kappa = 1$ (diamond line), $\kappa = 5$ (solid line), $\kappa = 10$ (circle line), $\kappa = 50$ (dashed line), and $\kappa = 100$ (boxed line).

THEOREM 5 (Novel population sizing): *Let κ be the number of desired niches, p_κ the probability of the least-fit niche's presence at equilibrium, and $\gamma := \Pr(B_1 > 0, \ldots, B_\kappa > 0)$ the desired joint niche presence probability. The novel model's population size n_N is given by:*

$$n_\mathrm{N} \geq \frac{\ln(1 - \sqrt[\kappa]{\gamma})}{\ln(1 - p_\kappa)}. \tag{50}$$

This result gives a lower bound $\ln(1 - \sqrt[\kappa]{\gamma})/\ln(1 - p_\kappa)$ for population size n_N necessary to obtain, with probabilities $0 < \gamma, p_\kappa < 1$, nonzero counts for the κ highest-fit niches.

If we take n as the independent variable in Equation (49), there are two other main parameters, namely κ and p_κ. In Figure 5, we independently investigate the effect of varying each of these. This figure clearly illustrates the risk of using too small population sizes n, an effect that has been demonstrated in experiments (Singh and Deb, 2006). For example, for $\kappa = 100$ and $n \approx 300$, we see from Figure 5 that the probability of all $\kappa = 100$ niches being present is essentially zero. Since their time complexity is $O(n)$, crowding algorithms can afford relatively large population sizes n; this figure illustrates the importance of doing so.

It is instructive to compare our novel population result with the following result from earlier work (Mahfoud, 1995, p. 175).

THEOREM 6 (Classical population sizing): *Let $r := f_\mathrm{min}/f_\mathrm{max}$ be the ratio between minimal and maximal of fitness optima in the κ niches; g the number of generations; and γ the probability of maintaining all κ niches. The population size n_C is given by*

$$n_\mathrm{C} = \left\lceil \frac{\kappa}{r} \left(-\ln \left(\frac{1 - \gamma^{\frac{1}{g}}}{\kappa} \right) \right) \right\rceil. \tag{51}$$

```
boolean PortfolioReplacement(f(p), f(c), generation)
begin
    r ← RandomDouble(0, 1)
    R ← R_i in the 2-tuple (w_i, R_i) ∈ 𝕎 such that w_{i-1} < r ≤ w_i {We put w_0 := 0}
    q ← R(f(p), f(c), generation) {Invoke replacement rule R from portfolio 𝕎}
    return q
end
```

Figure 6: The portfolio replacement rule, which combines different replacement rules. For example, it can combine the Deterministic crowding rule, the Probabilistic crowding rule, the Metropolis rule, and the Boltzmann rule.

We note that Equation (51) is based on considering selection alone. Here, the two only possible outcomes are that an existing niche is maintained, or an existing niche is lost (Mahfoud, 1995, p. 175). In order for successful niche maintenance to occur, it is required that the κ niches are maintained for all g generations. This is reflected in Equation (51) as follows. When the number of generations g increases, the expression $\gamma^{\frac{1}{g}}$ will get closer to one, and the required population size n_C will increase as a result. This reflects the fact that with selection only operating, niches can only be lost.

One could argue that the focus on loss only is appropriate for DeterministicReplacement but too conservative for ProbabilisticReplacement, since under this latter scheme niches may be lost, but they may also be gained. When inspecting the last generation's population, say, one is interested in whether a representative for a niche is there or not, and not whether it had been lost previously. In Equation (51), using g with the actual number of generations run can be used to give a conservative population sizing estimate, while setting $g = 1$ gives a less conservative population sizing estimate. Both of these approaches are investigated in Section 8.

6 Portfolios of Replacement Rules in Crowding

From our analytical result in Section 5, a reader might expect that deterministic crowding could give too strong convergence, while probabilistic crowding could give too weak convergence. Is there a middle ground?

To answer this question, we present the portfolio replacement rule PortfolioReplacement (R_U), which was briefly introduced in Section 5. In this section we discuss PortfolioReplacement in detail and also show how it can be analyzed using generalizations of the approaches employed in Section 5. As an illustration, we combine deterministic and probabilistic crowding. We hypothesize that a GA practitioner might want to combine other replacement rules in a portfolio as well, in order to obtain better results than provided by using individual replacement rules on their own.

6.1 A Portfolio of Replacement Rules

PortfolioReplacement (R_U) is a novel replacement rule which generalizes the replacement rules described in Section 4.3 by relying on a portfolio of (atomic) replacement rules. Under the PortfolioReplacement rule, which is presented in Figure 6, a choice is made from a set, or a portfolio, of replacement rules. Each replacement rule is chosen with a certain probability. The choice is based on a probability associated with each replacement rule as follows.

DEFINITION 7 (Replacement rule portfolio): *A replacement rule portfolio* $\mathbb{R}$ *is a set of* q *2-tuples*

$$\mathbb{R} = \{(p_1, R_1), \ldots, (p_q, R_q)\}$$

where $\sum_{i=1}^{q} p_i = 1$ *and* $0 \leq p_i \leq 1$ *for all* $1 \leq i \leq q$.

In Definition 7, and for $1 \leq i \leq q$, (p_i, R_i) means that the i-th replacement rule R_i is picked and executed with probability p_i when a rule is selected from $\mathbb{R}$ by the crowding GA. An equivalent alternative to $\mathbb{R}$, used in PORTFOLIOREPLACEMENT, is the cumulative (replacement) rule portfolio $\mathbb{W}$, defined as follows:

$$\mathbb{W} = \left\{ \left(\sum_{i=1}^{1} p_i, R_1 \right), \ldots, \left(\sum_{i=1}^{q} p_i, R_q \right) \right\} = \{(w_1, R_1), \ldots, (w_q, R_q)\}. \tag{52}$$

When invoked with the parameter $R =$ PORTFOLIOREPLACEMENT, the CROWDINGGA chooses among all the replacement rules included in the portfolio $\mathbb{W}$ for that invocation of the GA. In Figure 6, we assume that $\mathbb{W}$ is defined according to Equation (52).

The PORTFOLIOREPLACEMENT replacement rule approach gives greater flexibility and extensibility than what has previously been reported for crowding algorithms. As an illustration, here are a few example portfolios.

Example 8 The portfolio $\mathbb{R} = \{(1, R_P)\}$ gives probabilistic crowding, while $\mathbb{R} = \{(1, R_D)\}$ gives deterministic crowding. The portfolio $\mathbb{R} = \{(\frac{1}{2}, R_D), (\frac{1}{2}, R_P)\}$ gives a balanced mixture of deterministic crowding and probabilistic crowding.

6.2 Analysis of the Portfolio Approach

We assume two niches $\mathbb{X}$ and $\mathbb{Y}$. For the portfolio approach, Equation (10) is generalized to include the crowding algorithm's random selection of a replacement rule R_i as follows:

$$\Pr(\boldsymbol{p} \in \mathbb{X}) = \sum_{(p_i, R_i) \in \mathbb{R}} \sum_{\mathbb{A}, \mathbb{B} \in \{\mathbb{X}, \mathbb{Y}\}} \Pr(\boldsymbol{w} \in \mathbb{X},\ \boldsymbol{p} \in \mathbb{A}, \boldsymbol{c} \in \mathbb{B}, R = R_i). \tag{53}$$

Using Bayes' rule and the independence of rule selection from $\mathbb{R}$ gives

$$\begin{aligned} &\Pr(\boldsymbol{w} \in \mathbb{X},\ \boldsymbol{p} \in \mathbb{A}, \boldsymbol{c} \in \mathbb{B}, R = R_i) \\ &\quad = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{A}, \boldsymbol{c} \in \mathbb{B}, R = R_i) \Pr(\boldsymbol{p} \in \mathbb{A}, \boldsymbol{c} \in \mathbb{B}) \Pr(R = R_i). \end{aligned}$$

Consequently, in the replacement phase of a crowding GA we now need to consider the full portfolio $\mathbb{R}$. For example, Equation (12) generalizes to

$$\begin{aligned} &\Pr(\boldsymbol{w} \in \mathbb{X},\ \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_i) \\ &\quad = \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_i) \Pr(\boldsymbol{c} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{X}) \Pr(\boldsymbol{p} \in \mathbb{X}) \Pr(R = R_i). \end{aligned}$$

Here, the new factors compared to those of the corresponding non-portfolio expression in Equation (12) are $\Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_i)$ and $\Pr(R = R_i)$; hence we focus on these and similar factors in the rest of this section.

For an arbitrary number of replacement rules in $\mathbb{R}$, the resulting winning probabilities for p_x and p_y for niches $\mathbb{X}$ and $\mathbb{Y}$ respectively are as follows:

$$p_x = \sum_{(p_i, R_i) \in \mathbb{R}} \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_i) p_i \tag{54}$$

$$p_y = \sum_{(p_i, R_i) \in \mathbb{R}} \Pr(\boldsymbol{w} \in \mathbb{Y} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_i) p_i. \tag{55}$$

More than two niches can easily be accommodated. Much of the analysis earlier in this section remains very similar due to Equation (53) and its Bayesian decomposition. One just needs to plug in new values, such as for p_x and p_y above in Equation (54) and Equation (55), to reflect the particular portfolio $\mathbb{R}$.

6.3 Combining Deterministic and Probabilistic Crowding Using a Portfolio

For probabilistic crowding, a challenge may arise with "flat" fitness functions with small differences between fitness values and corresponding mild selection pressure. Such fitness functions can be tackled by means of our portfolio approach, and in particular by combining deterministic and probabilistic crowding.

Consider the portfolio $\mathbb{R} = \{(p_{\mathrm{D}}, R_{\mathrm{D}}), (p_{\mathrm{P}}, R_{\mathrm{P}})\}$, with $p_{\mathrm{D}} + p_{\mathrm{P}} = 1$, and suppose that the setup is as described in Section 5.1, namely two niches $\mathbb{X}$ and $\mathbb{Y}$ with the same probability of transitioning between them. Let us further assume that $f(\boldsymbol{x}) < f(\boldsymbol{y})$. At equilibrium we have $X = p_x$, see Equation (21). Now, p_x needs to reflect that two different replacement rules are being used in $\mathbb{R}$ or $\mathbb{W}$ when determining the winning probability $\Pr(\boldsymbol{w} \in \mathbb{X})$, say. To do so, we condition also on the random variable R representing the GA's randomly selected replacement rule and use the law of total probability:

$$\begin{aligned} p_x = {} & \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_{\mathrm{D}}) \Pr(R = R_{\mathrm{D}}) \\ & + \Pr(\boldsymbol{w} \in \mathbb{X} \mid \boldsymbol{p} \in \mathbb{X}, \boldsymbol{c} \in \mathbb{Y}, R = R_{\mathrm{P}}) \Pr(R = R_{\mathrm{P}}) \end{aligned}$$

which simplifies as follows

$$p_x = p_{\mathrm{P}} \times \frac{f(\boldsymbol{x})}{f(\boldsymbol{x}) + f(\boldsymbol{y})}. \tag{56}$$

Along similar lines, we obtain for $\mathbb{Y}$:

$$p_y = p_{\mathrm{D}} + p_{\mathrm{P}} \times \frac{f(\boldsymbol{y})}{f(\boldsymbol{x}) + f(\boldsymbol{y})}. \tag{57}$$

Here, p_{D} and p_{P} are the "knobs" used to control the GA's performance. When $p_{\mathrm{D}} \to 1$ one approaches pure deterministic crowding, while when $p_{\mathrm{P}} \to 1$ one approaches pure probabilistic crowding. The optimal settings of p_{D} and p_{P}, used to fruitfully combine deterministic and probabilistic crowding, depend on the application and the fitness function at hand.

Here is an example of a "flat" fitness function.

Example 9 Let $f_1(\boldsymbol{x}) = \sin^6(5\pi \boldsymbol{x})$ (see also Section 8.2) and define $f_3(\boldsymbol{x}) = f_1(\boldsymbol{x}) + 1000$.[6] Consider the portfolio $\mathbb{R} = \{(p_D, R_D), (p_P, R_P)\}$. Suppose that we have individuals $\boldsymbol{x}$ and $\boldsymbol{y}$ with $f_3(\boldsymbol{y}) = 1001$ and $f_3(\boldsymbol{x}) = 1000$. Using the portfolio approach in Equation (57) with $p_D = 0.9$ and $p_P = 0.1$, we obtain this probability $p_{\boldsymbol{x}}$ for $\boldsymbol{y}$ winning over $\boldsymbol{x}$:

$$p_{\boldsymbol{y}} = 0.9 + 0.1 \times \frac{f_3(\boldsymbol{y})}{f_3(\boldsymbol{x}) + f_3(\boldsymbol{y})} \approx 0.95.$$

In contrast, with pure probabilistic crowding ($p_D = 0$ and $p_P = 1$) we obtain

$$p_{\boldsymbol{y}} = \frac{f_3(\boldsymbol{y})}{f_3(\boldsymbol{x}) + f_3(\boldsymbol{y})} \approx 0.5.$$

This example illustrates the following general point: The flatter the fitness function, the greater the probability p_D (and the smaller the probability p_P) should be in order to obtain a reasonably high winning probability $p_{\boldsymbol{y}}$ for a better-fit niche such as $\mathbb{Y}$.

7 A Markov Chain Perspective

We now discuss previous analysis of genetic and stochastic local search algorithms using Markov chains (Goldberg and Segrest, 1987; Nix and Vose, 1992; Harik et al., 1997; De Jong and Spears, 1997; Spears and De Jong, 1997; Cantu-Paz, 2000; Hoos, 2002; Moey and Rowe, 2004a,b; Mengshoel, 2008). In addition, we discuss how our analysis in Section 5 and Section 6 relates to these previous analysis efforts.

7.1 Markov Chains in Genetic Algorithms

Most evolutionary algorithms simulate a Markov chain in which each state represents one particular population. For example, consider the simple genetic algorithm (SGA) with fixed-length bitstrings, one-point crossover, mutation using bit-flipping, and proportional selection. The SGA simulates a Markov chain with $|\mathcal{S}| = \binom{n+2^m-1}{2^m-1}$ states, where n is the population size and m is the bitstring length (Nix and Vose, 1992). For nontrivial values of n and m, the large size of $\mathcal{S}$ makes exact analysis difficult. In addition to the use of Markov chains in SGA analysis (Goldberg and Segrest, 1987; Nix and Vose, 1992; Suzuki, 1995; Spears and De Jong, 1997), they have also been applied to parallel genetic algorithms (Cantu-Paz, 2000). Markov chain lumping or state aggregation techniques, to reduce the problem of exponentially large state spaces, have been investigated as well (De Jong and Spears, 1997; Spears and De Jong, 1997; Moey and Rowe, 2004a).

It is important to note that most previous work has been on the SGA (Goldberg and Segrest, 1987; Nix and Vose, 1992; Suzuki, 1995; Spears and De Jong, 1997), not in our area of niching or crowding GAs. Further, much previous work has used an exact but intractable Markov chain approach, while we aim for inexact but tractable analysis in this article.

[6] An anonymous reviewer is acknowledged for suggesting this example.

7.2 Markov Chains in Stochastic Local Search

For stochastic local search (SLS) algorithms using bit-flipping, the underlying search space is a Markov chain that is a hypercube. Each hypercube state $\boldsymbol{x} \in \{0,1\}^m$ represents a bitstring. Each state $\boldsymbol{x}$ has m neighbors, namely those states that are bitstrings one flip away from $\boldsymbol{x}$. As search takes place in a state space $\mathcal{S} = \{\boldsymbol{b} \mid \boldsymbol{b} \in \{0,1\}^m\}$, with size $|\mathcal{S}| = 2^m$, analysis can also be done in this space. However, such analysis is costly for nontrivial values of m since the size of $\mathcal{P}$ is $|\{0,1\}^m| \times |\{0,1\}^m| = 2^{m+1}$ and the size of $\mathcal{V}$ is $|\{0,1\}^m| = 2^m$.

In related research, we have introduced two approximate models for SLS, the naive and trap Markov chain models (Mengshoel, 2008). Extending previous research (Hoos, 2002), these models improve the understanding of SLS by means of expected hitting time analysis. Naive Markov chain models approximate the search space of an SLS by using three states. Trap Markov chain models extend the naive models by (i) explicitly representing noise and (ii) using state spaces that are larger than those of naive Markov chain models but smaller than the corresponding exact models.

Trap Markov chains are related to the simple and branched Markov chain models introduced by Hoos (2002). Hoos' Markov chain models capture similar phenomena to our trap Markov chains, but the latter have a few novel and important features. First, a trap Markov chain has a noise parameter, which is essential when analyzing the impact of noise on SLS. Second, while it is based on empirical considerations, the trap Markov chain approach is derived analytically based on work by Deb and Goldberg (1993).

7.3 Our Analysis

Our analysis in Section 5 and Section 6 is related to a Markov chain analysis as follows. In CROWDINGSTEP (see Figure 3), MATCH creates a matching $\boldsymbol{m}^*$. This matching step is followed by local tournaments, each between a child $\boldsymbol{c}$ and a parent $\boldsymbol{p}$, where $(\boldsymbol{c}, \boldsymbol{p}) \in \boldsymbol{m}^*$, with outcome newPop[$k$] $\leftarrow \boldsymbol{c}$ or newPop[k] $\leftarrow \boldsymbol{p}$. For each population array location newPop[k], one can introduce a Markov chain. The state of each Markov chain represents the niche of a parent, and probabilities on transitions represent the corresponding probabilities of outcomes of local tournaments for newPop[k] between the parent $\boldsymbol{p}$ and the matched child $\boldsymbol{c}$.

Our approach is most similar to previous research using aggregated Markov chain states in evolutionary algorithms and stochastic local search (De Jong and Spears, 1997; Spears and De Jong, 1997; Hoos, 2002; Moey and Rowe, 2004a,b; Mengshoel, 2008). Such aggregation is often possible with minimal loss of accuracy (Moey and Rowe, 2004a,b; Mengshoel, 2008). In a crowding GA Markov chain model, each position in the GA's population array can be associated with a Markov chain. However, Markov chain transitions are not restricted to one-bit flips (as they typically are in SLS), but depend on factors such as tournament size S, crossover probability P_{C}, and mutation probability P_{M}. With large S, small P_{C}, and small P_{M}, tournaments clearly will be very local (in other words between individuals with small genotypic distance). On the other hand, given small S, large P_{C}, and large P_{M}, tournaments clearly will be less local (in other words between individuals with larger genotypic distance). A detailed investigation of the interaction between these different parameters from a Markov chain perspective is an interesting direction for future research.

8 Experiments

In order to complement the algorithms and theoretical framework developed so far in this article, we report on experimental results in this section. We have experimented with our crowding approach—and in particular ProbabilisticCrowding—under progressively more challenging conditions as follows. As reported in Section 8.1, we first used the SimpleStep crowding algorithm and its idealized variation operator along with quite simple fitness functions. The remaining sections employed the GeneralPCGA. Section 8.2 presents results obtained using the CrowdingStep algorithm which uses traditional crossover and mutation along with classical fitness functions. Finally, in Section 8.3 we present empirical population sizing results; again the CrowdingStep algorithm was used.

8.1 Experiments Using Idealized Operators and SimpleStep

The purposes of these experiments were to: (i) Check whether the deterministic difference equation analysis models the stochastic situation in experiments well; and (ii) Check whether the approach of picking a candidate from each niche is reasonable in the analysis. In order to achieve these goals, we used the SimpleStep algorithm as well as quite large population sizes. These initial experiments were performed using a fitness function with only q discrete niches (each of size one) and mutation probability p_ℓ idealized as uniform jump probability to one of the other niches. The probabilistic crowding replacement rule R_{P} was used to choose the winner in a tournament.

8.1.1 Two Niches, Same Jump Probabilities

In our first experiment, using the SimpleStep algorithm and the ProbabilisticReplacement rule, we consider two niches: $\mathbb{X} = \{0\}$ and $\mathbb{Y} = \{1\}$ and the fitness function $f_3(0) = 1$, $f_3(1) = 4$. Since there were two niches, the solutions to the difference equations in Equation (27) can be applied along with $p_{\boldsymbol{x}} = f(\boldsymbol{x})/(f(\boldsymbol{x}) + f(\boldsymbol{y})) = f_3(0)/(f_3(0) + f_3(1)) = 1/5$ and $p_{\boldsymbol{y}} = f(\boldsymbol{y})/(f(\boldsymbol{x}) + f(\boldsymbol{y})) = f_3(1)/(f_3(1) + f_3(0)) = 4/5$. This gives the following niche proportions:

$$E(X(t)) = \frac{1}{5} + \left(\frac{1}{2} - \frac{1}{5}\right) p_{\mathrm{s}}^t = \frac{1}{5} + \frac{3}{10} p_{\mathrm{s}}^t \tag{58}$$

and

$$E(Y(t)) = \frac{4}{5} + \left(\frac{1}{2} - \frac{4}{5}\right) p_{\mathrm{s}}^t = \frac{4}{5} - \frac{3}{10} p_{\mathrm{s}}^t. \tag{59}$$

We let $p_{\mathrm{s}} = 0.8$, used population size $n = 100$, and let the crowding GA run for 50 generations. (A variation probability $p_\ell = 1 - p_{\mathrm{s}} = 0.2$ might seem high, but recall that this operation gives jumps between niches, and is not the traditional bit-wise mutation operator.) A plot of experimental versus predicted results for both niches is provided in Figure 7. Using Equations (58) and (59), we obtain respectively $\lim_{t\to\infty} E(X(t)) = \frac{1}{5}$, hence $\mu_x = \frac{1}{5} \times 100 = 20$, and $\lim_{t\to\infty} E(Y(t)) = \frac{4}{5}$, hence $\mu_y = \frac{4}{5} \times 100 = 80$. Alternatively, and using the approach of Section 5, we obtain $\mu_{\boldsymbol{x}} = np_{\boldsymbol{x}} = 100 \times \frac{1}{5} = 20$, $\sigma_{\boldsymbol{x}}^2 = np_{\boldsymbol{x}}(1 - p_{\boldsymbol{x}}) = 16$ and $\mu_{\boldsymbol{y}} = np_{\boldsymbol{y}} = 100 \times \frac{4}{5} = 80, \sigma_{\boldsymbol{y}}^2 = np_{\boldsymbol{y}}(1 - p_{\boldsymbol{y}}) = 16$. In the figure, we notice that the experimental results follow these predictions quite well. In general, the

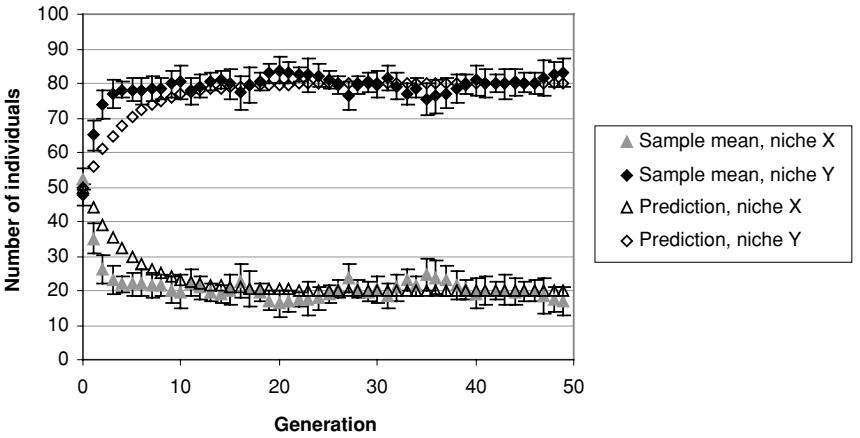

Figure 7: Predicted results versus experimental results for probabilistic crowding, using a simple fitness function f_3 with two niches $\mathbb{X} = \{0\}$ and $\mathbb{Y} = \{1\}$. The fitness function is $f_3(0) = 1$ and $f_3(1) = 4$. Here we show empirical results, including 95% confidence intervals, for both $\mathbb{X}$ and $\mathbb{Y}$ averaged over 10 runs with different initial populations.

predictions are inside their respective confidence intervals. There is some noise, but this is as expected, since a GA with a probabilistic replacement rule is used.

8.1.2 Multiple Niches, Same Jump Probabilities

In the second experiment, the SIMPLESTEP algorithm and the PROBABILISTICREPLACEMENT rule were again used. The fitness function $f_4(\boldsymbol{x}_i) = i$, for integer $1 \leq i \leq 8$, gave $q = 8$ niches. Here we can use Equation (44) with $\xi_{ji} = 1$, giving

$$p_i = \frac{f(\boldsymbol{x}_i)}{\sum_{i=1}^{q} f(\boldsymbol{x}_i)} \tag{60}$$

with, for example, $p_1 = 1/36$, $p_4 = 4/36$, and $p_8 = 8/36$ for niches $\mathbb{X}_1$, $\mathbb{X}_4$, and $\mathbb{X}_8$, respectively.

A population size of $n = 360$ was used in our experiments, and the GA was run for $g_{\mathrm{N}} = 50$ generations. With the probabilities just mentioned for $\mathbb{X}_1$, $\mathbb{X}_4$, and $\mathbb{X}_8$, we get predicted subpopulation sizes $np_1 = 10$, $np_4 = 40$, and $np_8 = 80$. A plot of experimental versus predicted results for $p_s = 0.8$ is provided in Figure 8. After a short initialization phase, the empirical results follow the predicted equilibrium results very well, although there is a certain level of noise also in this case, as expected. In the majority of cases, the predicted mean is inside the confidence interval of the sample mean. Qualitatively, it is important to notice that all the niches, even $\mathbb{X}_1$, are maintained reliably.

An analysis of the amount of noise can be performed as follows, using results from Section 5. As an example, for niche $\mathbb{X}_1$ we obtain $\sigma_1^2 = 360 \times \frac{1}{36} \times \frac{35}{36}$, which gives $\sigma_1 \approx 3.1$. For $\mathbb{X}_4$ and $\mathbb{X}_8$ we similarly get $\sigma_4 \approx 6.0$ and $\sigma_8 \approx 7.9$ respectively. The fact that the observed noise increases with the fitness of a niche, as reflected in corresponding increases in lengths of confidence intervals in Figure 8, is therefore in line with our analytical results.

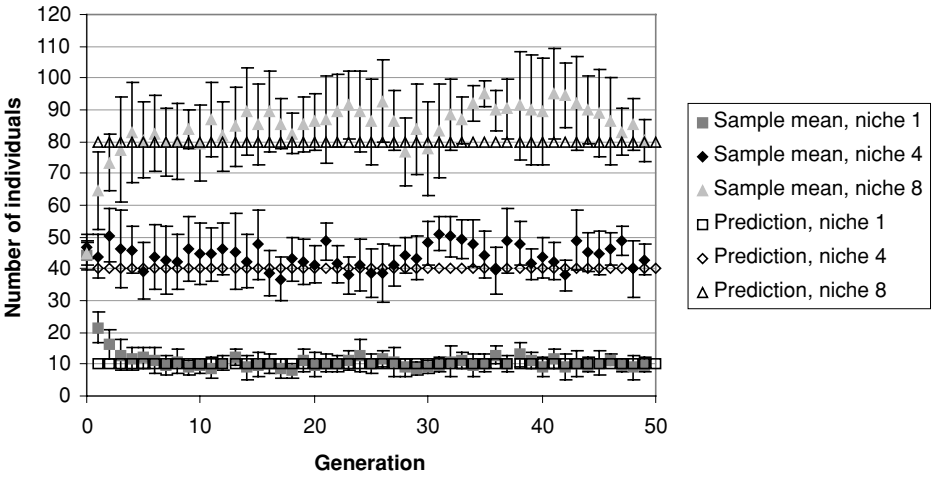

Figure 8: Predicted versus experimental results for fitness function f_4's niches $\mathbb{X}_1$, $\mathbb{X}_4$, and $\mathbb{X}_8$. The predicted results are based on steady-state expected number of individuals in different niches in the population. The experimental results are sample means for 10 runs (with different initial populations) and include 95% confidence intervals.

8.2 Experiments Using Traditional Operators and CrowdingStep

In this section, we report on the empirical investigations of the CrowdingStep algorithm, which uses traditional mutation and crossover operators. Experiments were performed using discretized variants of the f_1 and f_2 test functions (Goldberg and Richardson, 1987), where

$$f_1(x) = \sin^6(5\pi x)$$
$$f_2(x) = e^{-2(\ln 2)\left(\frac{x-0.1}{0.8}\right)^2} \sin^6(5\pi x).$$

These two functions are of interest for a number of reasons. First, they have multiple local optima, and are therefore representative of certain applications in which multimodality is found. In f_1, all local optima are global optima. In f_2, the magnitude of the local optima decreases with increasing x. Second, these functions have been used as test functions in previous research on niching algorithms (Goldberg and Richardson, 1987; Yin, 1993; Harik, 1995; Goldberg and Wang, 1998; Singh and Deb, 2006).

In our analysis of the experiments, the domain [0, 1] of the two test functions was split up into 25 equally-sized subintervals $[a, b)$ or $[a, b]$. For each test function, predicted allocation and experimental allocation of individuals were considered. Predicted allocation, in terms of individuals in an interval $[a, b)$ or $[a, b]$, was computed by forming

$$n \int_a^b f_i(x)\,\mathrm{d}x \Big/ \int_0^1 f_i(x)\,\mathrm{d}x.$$

This prediction is closely related to the niching rule. Experimental allocation is merely the observed number of individuals in the interval $[a, b)$ or $[a, b]$, averaged over the number of experiments performed.

For f_1 we show experimental results for two variants of the probabilistic crowding algorithm GeneralPCGA. In the mutation only variant (the M variant) we used $P_C = 0$, $P_M = 0.1$, $n = 200$, and $g_N = 100$. For the variant using both mutation and crossover (the M+C variant), we used $P_C = 1.0$ and $P_M = 0.3$; n and g_N were the same.

The behavior of variant M+C is illustrated in Figure 9. This figure shows, using the f_1 test function, how niches emerge and are maintained by the algorithm in one experimental run. The main result for f_1 is that the probabilistic crowding variants M and M+C give a reliable niching effect as desired. The five global maxima emerge early and are, in general, reliably maintained throughout an experiment. The allocation of individuals in the population reflects the shape of f_1 quite early; there is some increase in peakiness with increasing generations.

The performance of GeneralPCGA on f_1 is presented in Figure 10. Before discussing these results in more detail, we distinguish between inter- and intra-niche effects as it relates to the allocation of individuals. Inter-niche effects take place between niches, while intra-niche effects happen inside a niche. Compared to the niching rule prediction, the main intra-niche effect observed in Figure 10 is that individuals close to optima are slightly over-represented at the expense of individuals farther away. For both GA variants in Figure 10, examples of this effect can be seen for the intervals [0.28–0.32) and [0.68–0.72). There are also inter-niche effects, in particular the fourth optimum from the left is over-sampled for both M and M+C variants. For the M variant, this is partly due to a few high-allocation outlying experiments as illustrated in (i) the wide confidence interval, and (ii) the difference between the sample mean and the sample median.

We now turn to f_2 and the experimental results for our two variants of GeneralPCGA. For the M variant, we used $P_C = 0$, $P_M = 0.1$, $n = 200$, and $g_N = 100$. For the M+C variant we used $P_C = 1.0$ and $P_M = 0.3$; n and g_N were the same.

Figure 11 illustrates the performance using the f_2 test function. The main result for f_2 is that the variants M and M+C give reliable niching as desired. Again, the maxima emerge early and are in general maintained reliably throughout an experiment. Intra-niche effects are for f_2 similar to those for f_1; in addition, our experiments suggest some inter-niche effects that are probably due to f_2's shape and go beyond those discussed for f_1. Specifically, compared to the prediction based on the niching rule, there is some over-sampling of the two greater local optima (to the left in Figure 11) at the expense of the two smaller local optima (to the right in Figure 11). This over-sampling is less pronounced for the M variant compared to the M+C variant, however.

Overall, our experiments show reliable niche maintenance and suggest that Equation (44), and in particular its special case the niching rule, can be applied also when the classical GA operators of GeneralPCGA are used. At least, our experiments suggest this for fitness functions that are similar in form to f_1 and f_2. Experimentally, we have found a slight over-sampling of higher-fit individuals compared to lower-fit individuals. Clearly, this over-sampling effect can have several underlying causes, including the noise induced by the GA's sampling, discretization of the underlying continuous functions into binary strings over which search is performed, and limitations of our analytical models. We leave further investigation of this issue as a topic for future research.

8.3 Population Sizing Experiments and CrowdingStep

Here we show how the population sizing results in Section 5.4 can be used, and provide experimental verification by means of the CrowdingStep algorithm. Specifically, we

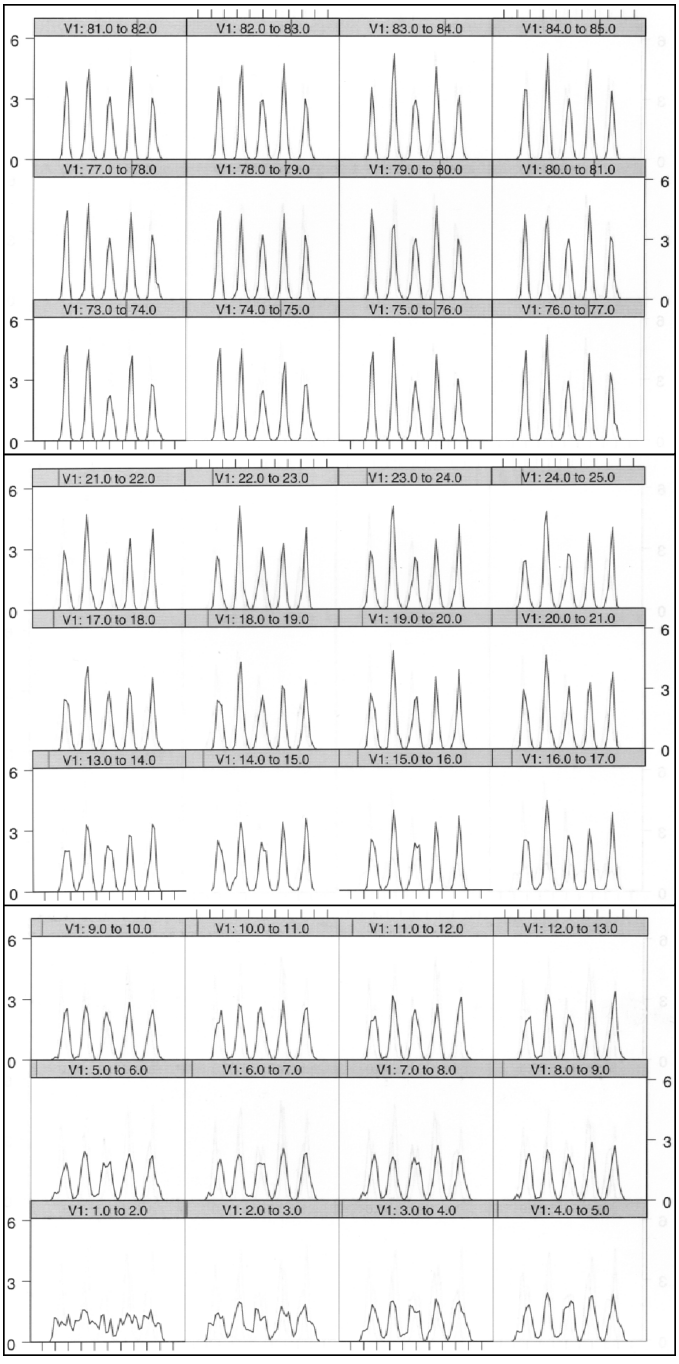

Figure 9: Probabilistic crowding variant GENERALPCGA with $n = 200$, $P_C = 1$, $P_M = 0.3$, and $g_N = 120$. The test function used is f_1, with the number of individuals on the y-axis. Generations are shown in increasing order from bottom left to top right. The bottom panel shows generations 1 through 12; the middle panel generations 13 through 24; and the top panel generations 73 through 84 (which are representative for later generations).

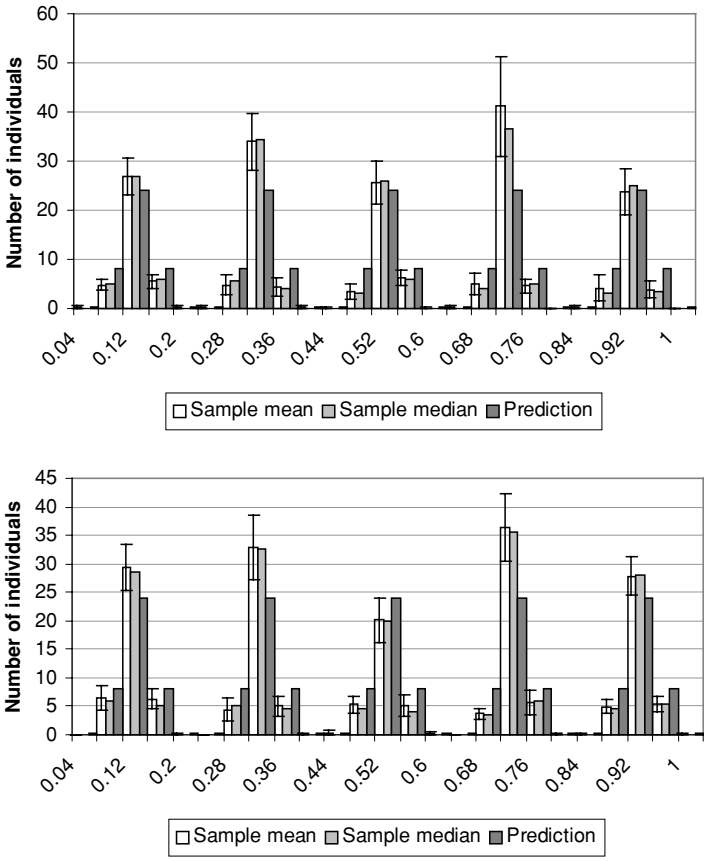

Figure 10: The performance of probabilistic crowding algorithm GENERALPCGA on the f_1 fitness function. We show empirical results, averaged over 10 experiments, at generation 100 for the case of mutation only (the M variant) at the top; for the case of both mutation and crossover (the M+C variant) at the bottom.

consider the f_4 fitness function used in Section 8.1.2. Suppose that we want to reliably maintain the three best fit niches $\mathbb{X}_6$, $\mathbb{X}_7$, and $\mathbb{X}_8$. For population sizing purposes, we need to consider all of these niches.

We used Equation (51) for population sizing as follows: Given known parameters κ, r, γ, and g, we computed the population size n. Two different approaches were used to set g. We either set $g = 1$ or $g = 50$, since 50 generations were used in the experiments here. The first setting $g = 1$ corresponds to essentially ignoring the effect of niche loss over generations. The second setting $g = 50$ is more conservative, and takes niche loss into account as discussed in Section 5.4. Using Equation (51), the conservative population size (using $g = 50$) is $n_C = 27$, while the less conservative population size ($g = 1$) is $n_C = 11$. Observed γ_o is computed as follows. For desired γ, r_e experimental runs were performed. Runs in which the top κ niches did not have a representative were counted, resulting in a value for failure runs r_f. These are runs where, at the last generation, at least one of the $\mathbb{X}_6$, $\mathbb{X}_7$, or $\mathbb{X}_8$ niches did not have a representative. Finally,

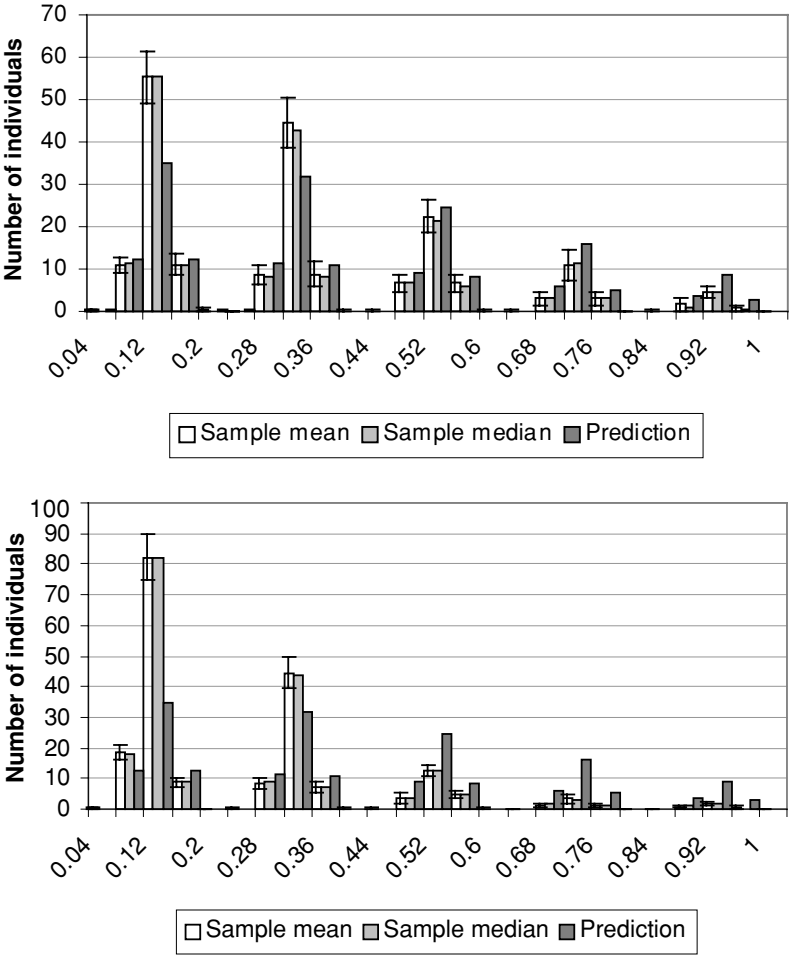

Figure 11: The performance of the probabilistic crowding algorithm GENERALPCGA on the f_2 fitness function. We show empirical results, averaged over 10 experiments, at generation 100 for the case of mutation only (the M variant) at the top; for the case of both mutation and crossover (the M+C variant) at the bottom.

Table 2: Population Size Predicted from Desired Reliability, Along with Observed Reliability

	Less conservative (set $g = 1$)		Conservative (set $g = 50$)	
Desired γ	Population size	Observed γ_o	Population size	Observed γ_o
0.80	11	0.74	27	1.0
0.95	17	0.93	32	1.0

the observed γ_o was computed as $\gamma_o = (r_e - r_f)/r_e$. In the experiments reported here, $r_e = 100$ was used.

In Table 2, the results from using the population sizing Equation (51) are summarized. We see that the less conservative population sizing approach is in closer

Table 3: Population Sizing Results for Probabilistic Crowding for the f_1 and f_2 Functions. The Population Sizes for the Classical and Novel Population Sizing Models as Shown for Each of the Test Functions f_1 and f_2.

Probability	**Population Size,** f_1		**Population Size,** f_2	
γ	Classical; n_C	Novel; n_N	Classical; n_C	Novel; n_N
0.9	20	18	79	50
0.99	32	28	125	79
0.999	43	39	171	109
0.9999	55	49	217	138
0.99999	66	59	263	167
0.999999	78	70	309	197

correspondence to the empirical data than the more conservative population sizing approach. This shows that the assumption of niche loss, at least for this particular test function, might be overly conservative.

Additional population sizing results for f_1 and f_2 are shown in Table 3. We note that the classical approach is intended as a model for deterministic crowding (before convergence), while the novel approach is a model of probabilistic crowding (after convergence). With those differences in mind, the results are quite similar, and it is not surprising that the classical approach gives more conservative results than our novel approach, especially for f_2 where the fitness ratio $r = f_{\min}/f_{\max}$ is four times greater than what r is for f_1. This ratio difference impacts the classical approach more than our novel approach.

9 Conclusion and Future Work

Inspired by multimodal fitness functions and deterministic crowding (Mahfoud, 1995), we have investigated crowding in genetic algorithms, and in particular the probabilistic crowding approach. Probabilistic crowding is a tournament selection algorithm using distance-based tournaments, and it employs a probabilistic rather than a deterministic acceptance function as basis for replacement. The two core ideas in probabilistic crowding are to (i) hold pairwise tournaments between bitstrings (or individuals) with small distance, and (ii) employ probabilistic tournaments. These two principles lead to a niching algorithm that is simple, predictable, and fast. In fact, our approach is an example instantiation of an algorithmic framework that supports different crowding algorithms, including different replacement rules and the use of multiple replacement rules in a portfolio. We have shown, analytically and experimentally, that our approach gives stable, predictable convergence that approximates the niching rule, a gold standard for niching algorithms. We also introduced a novel, more general niching rule, that generalizes the niching rule known from previous research. In addition, a new population sizing result for crowding algorithms was provided.

This research also identifies probabilistic crowding as a member of a class of algorithms that we call local tournament algorithms. Local tournament algorithms also include deterministic crowding, restricted tournament selection, parallel recombinative simulated annealing, the Metropolis algorithm, and simulated annealing. By introducing portfolios of replacement rules, we have shown how replacement rules from different local tournament algorithms can be combined in a principled way. We

illustrated the benefit of using portfolios by combining deterministic and probabilistic crowding, thereby increasing performance on "flat" fitness functions.

Future work includes the following. First, experiments on harder fitness functions, such as complex Bayesian networks, would be interesting. Second, a more detailed Markov chain analysis could perhaps explain some of the intra- and inter-niche effects found in experiments. Third, it would be interesting to further explore our novel population sizing result, in order to more fully understand what it means for other niching algorithms.

Acknowledgments

Dr. Mengshoel's contribution to this work was in part sponsored by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, NRL Grant N00014-97-C-2061, and NASA Cooperative Agreement NCC2-1426. Professor Goldberg's contribution to this work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF under grants F49620-03-1-0129, AF9550-06-1-0096, and AF9550-06-1-0370. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the US Government.

References

Ando, S., Sakuma, J., and Kobayashi, S. (2005a). Adaptive isolation model using data clustering for multimodal function optimization. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-05)*, pages 1417–1424.

Ando, S., Suzuki, E., and Kobayashi, S. (2005b). Sample-based crowding method for multimodal optimization in continuous domain. In *The 2005 IEEE Congress on Evolutionary Computation*, pages 1867–1874.

Ballester, P. J. and Carter, J. N. (2003). Real-parameter genetic algorithms for finding multiple optimal solutions in multi-modal optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-03)*, pages 706–717.

Ballester, P. J. and Carter, J. N. (2004). An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-04)*, pages 901–913.

Ballester, P. J. and Carter, J. N. (2006). Characterising the parameter space of a highly nonlinear inverse problem. *Inverse Problems in Science and Engineering*, 14:171–191.

Cantu-Paz, E. (2000). Markov chain models of parallel genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(3):216–226.

Culberson, J. (1992). Genetic invariance: A new paradigm for genetic algorithm design. Technical Report 92-02, University of Alberta, Department of Computer Science, Alberta, Canada.

Darwen, P. and Yao, X. (1996). Every niching method has its niche: Fitness sharing and implicit sharing compared. In *Proceedings Parallel Problem Solving from Nature—PPSN IV*, pages 398–407. Springer, New York.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, NY.

Deb, K. and Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, D., editor, *Foundations of Genetic Algorithms II*, pages 93–108. Morgan Kaufmann, San Mateo, CA.

DeJong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor.

De Jong, K. A. and Spears, W. M. (1997). Analyzing GAs using Markov models with semantically ordered states. In *Foundations of Genetic Algorithms 4*, pages 85–100. Morgan Kaufmann, San Mateo, CA.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufman, San Mateo, CA.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA.

Goldberg, D. E. (1990). A note on Boltzmann tournament selection for genetic algorithms and population oriented simulated annealing. *Complex Systems*, 4:445–460.

Goldberg, D. E., Deb, K., and Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. Technical Report IlliGAL Report No 92005, University of Illinois, Urbana.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Erlbaum, Hillsdale, NJ.

Goldberg, D. E. and Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. In Grefenstette, J. J., editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 1–8. Erlbaum, Hillsdale, NJ.

Goldberg, D. E. and Wang, L. (1998). Adaptive niching via coevolutionary sharing. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 21–38. John Wiley and Sons, Chichester, UK.

Harik, G., Cantu-Paz, E., Goldberg, D. E., and Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 7–12.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, San Mateo, CA.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109.

Hocaoglu, C. and Sanderson, A. C. (1997). Multimodal function optimization using minimal representation size clustering and its application to planning multipaths. *Evolutionary Computation*, 5(1):81–104.

Hoos, H. H. (2002). A mixture-model for the behaviour of SLS algorithms for SAT. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 661–667.

Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Laarhoven, P. J. M. v. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications. Mathematics and Its Applications*. D. Reidel, Dordrecht, The Netherlands.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL. IlliGAL Report 95001.

Mahfoud, S. W. and Goldberg, D. E. (1995). Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing*, 21:1–28.

Mengshoel, O. J. (1999). *Efficient Bayesian Network Inference: Genetic Algorithms, Stochastic Local Search, and Abstraction*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Mengshoel, O. J. (2008). Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172: 955–990.

Mengshoel, O. J. and Goldberg, D. E. (1999). Probabilistic crowding: Deterministic crowding with probabilistic replacement. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 409–416.

Mengshoel, O. J. and Wilkins, D. C. (1998). Genetic algorithms for belief network inference: The role of scaling and niching. In *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 547–556.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092.

Moey, C. C. J. and Rowe, J. E. (2004a). Population aggregation based on fitness. *Natural Computing*, 3(1):5–19.

Moey, C. C. J. and Rowe, J. E. (2004b). A reduced Markov model of GAs without the exact transition matrix. In *Parallel Problem Solving from Nature (PPSN)*, pages 72–80.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, Canada.

Newman, M. E. J. and Barkema, G. T. (1999). *Monte Carlo Methods in Statistical Physics*. Oxford University Press, Oxford, UK.

Nix, A. E. and Vose, M. D. (1992). Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5(1):77–88.

Pelikan, M. and Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-01)*, pages 511–518.

Pétrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 798–803.

Sastry, K., Abbass, H. A., Goldberg, D. E., and Johnson, D. D. (2005). Sub-structural niching in estimation of distribution algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-05)*, pages 671–678.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343.

Singh, G. and Deb, K. (2006). Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-06)*, pages 1305–1312.

Spears, W. M. and De Jong, K. A. (1997). Analyzing GAs using Markov models with semantically ordered and lumped states. In Belew, R. K. and Vose, m. D., editors, *Foundations of Genetic Algorithms 4*, pages 85–100. Morgan Kaufmann, San Mateo, CA.

Suzuki, J. (1995). A Markov chain analysis on simple genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(5):655–659.

Thierens, D. and Goldberg, D. E. (1994). Elitist recombination: An integrated selection recombination GA. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 152–159.

Yin, X. (1993). A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufman, San Mateo, CA.