Complex Systems 4 (1990) 445–460

A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing

David E. Goldberg
University of Illinois at Urbana–Champaign, Urbana IL 61801 USA

Abstract. This note describes a selection procedure for genetic algorithms called *Boltzmann tournament selection*. As simulated annealing evolves a Boltzmann distribution in time using the Metropolis algorithm or a logistic acceptance mechanism, Boltzmann tournament selection evolves a Boltzmann distribution across a population and time using pairwise probabilistic acceptance and anti-acceptance mechanisms. In this note, the motivation, the theory of operation, some proof-of-principle computational experiments, and a Pascal implementation of the algorithm are presented. The efficient use of Boltzmann tournament selection on parallel hardware and its connection to other niching mechanisms are also considered.

1. Introduction

Simulated annealing (SA) [14] uses the notions of (1) a probabilistic acceptance rule, (2) exploration in the neighborhood of the current solution, and (3) the Boltzmann distribution and thermal equilibrium to guarantee asymptotic convergence to global optima in combinatorial optimization problems [1]. Genetic algorithms (GAs) have relied on (1) probabilistic selection rules (reproduction or selection), (2) exploration in the neighborhood of the current solution (mutation), and (3) recombination of parental solutions (crossover) to promote their search [7], but GAs have sometimes been criticized because they lack convergence proofs, asymptotic or otherwise. In this note, the notions of thermal equilibrium and the Boltzmann distribution are borrowed from simulated annealing and adapted to genetic algorithm practice. Specifically, a Boltzmann tournament selection procedure is derived and implemented to give stable distributions within a population of structures that are provably near Boltzmann. Using such a mechanism carries over to genetic algorithms the same guarantees of asymptotic convergence enjoyed by simulated annealing. It also creates another niching mechanism for forming and sizing stable subpopulations of individuals according to differences among them, if the cooling process is not taken to the limit.

Simulated annealing	Genetic algorithms
Metropolis or logistic acceptance	reproduction
neighborhood generation	mutation
cooling & Boltzmann distribution	???
???	recombination
???	expression
???	recoding
???	niching
???	speciation

Table 1: Comparison of simulated annealing to genetic algorithms.

In the remainder, the connections between simulated annealing and genetic algorithms are drawn more carefully. Thereafter, the Boltzmann tournament selection procedure is derived, analyzed, and implemented in the Pascal programming language. Proof-of-principle computations are performed to verify the theory, and the procedure is suggested for use in genetic algorithms as well as in parallel simulated annealing algorithms.

2. Simulated annealing and genetic algorithms

The connections between genetic algorithms and simulated annealing have been explored elsewhere, but only loosely. Davis and Steenstrup [2] acknowledge the procedures' common natural roots, but it is odd that this lead essay from a collection of papers entitled *Genetic Algorithms and Simulated Annealing* chooses only to describe each method separately, largely ignoring connections between them. Sirag and Weisser [17] get somewhat closer to the heart of the matter in their study of "thermally" motivated adaptation of mutation and crossover probabilities; however, tighter connections may be drawn in a straightforward manner.

Note that simulated annealing contains three basic elements:

1. probabilistic acceptance (Metropolis or logistic forms),
2. neighborhood exploration, and
3. a cooling schedule that respects thermal equilibrium.

On the other hand, genetic algorithms may consist of any number of different operators:

1. probabilistic selection (via reproduction or replacement),
2. recombination,
3. recoding (reordering),
4. expression,
5. diversity generation (mutation or other),

6. niching, and
7. speciation.

Comparing these lists, we may understand the lacunae of each method by placing related operators in rough similitude. This is done in table 1, where question marks are used to show the absence of a corresponding capability. GAs and SAs share the notions of acceptance (selection or reproduction) and generation of trial solutions in the neighborhood of the current solution; however, thereafter the similarity ends. Simulated annealing alone has the notions of thermal equilibrium and a cooling schedule,[1] while only genetic algorithms have focused on the importance of recombination and other operators found in living systems. The importance of recombination and other operators to the search of difficult nonlinear problems has been stressed elsewhere [7–10,13] and will not be repeated here. Instead, the notions of thermal equilibrium and cooling are adapted to population-oriented genetic search procedures. Carrying the idea over is not straightforward, and attempts to parallelize serial simulated annealing algorithms have not been overwhelmingly successful:

> Our general feeling is that the sequential nature of the simulated annealing algorithm in its present form is a drawback that greatly hampers the design of massively parallel simulated annealing algorithms. In our opinion, the design of such algorithms requires an approach that differs substantially from those presented in this chapter [1, p. 114].

This is true enough, but the cure suggested by these writers (Boltzmann machines used as function optimizers) misses the malady. The problem with much work in parallel simulated annealing is not that the method is hostile to parallelization, but rather that researchers have tried too hard to implement a serial SA in parallel. One of the lessons of genetic algorithm research is to adopt a populations viewpoint, and the selection mechanism I am about to propose was created with this view in mind. It is thus equally suitable for asynchronous selection in a parallel simulated annealing algorithm or for use in population-oriented genetic algorithms. Although the particular algorithm to be described is implemented serially, it may be easily adapted to parallel hardware because it only uses local information.

3. Boltzmann tournament selection

To motivate Boltzmann tournament selection, consider the steps in one variant of simulated annealing:

1. Generate a candidate solution uniformly at random in the neighborhood of the current solution.

[1]This lacking on the part of GAs has been partially made up through the use of niching mechanisms. The connection between Boltzmann distributions and niching will be made clearer in section 5.

2. Accept the new solution with logistic probability $p = \exp(-E_{\text{new}}/T)/[\exp(-E_{\text{new}}/T) + \exp(-E_{\text{old}}/T)]$, where the E values are the objective function values to be minimized and T is a control parameter analogous to temperature.

3. Repeat at step 1 for some number of trials.

The fundamental idea behind simulated annealing is that this generation-acceptance iteration is repeated often enough to guarantee that the temporal distribution of individuals becomes Boltzmann. Thereafter, the temperature is reduced gradually, keeping the distribution in equilibrium, thereby guaranteeing the discovery of globally optimal structures asymptotically as the temperature goes to zero. We will sidestep the knotty issue of cooling here, even though choice of an appropriate cooling schedule is important to the quality of solution ultimately attained and the speed with which it is achieved. Instead we focus solely on getting a near-Boltzmann distribution to form across a population as time goes on. Doing so will carry over the asymptotic convergence theorems of simulated annealing to genetic algorithms immediately.

The skeleton SA algorithm outlined above is different from the usual description, as it uses the logistic probability of acceptance rather than the Metropolis algorithm; at steady state, both achieve the desired Boltzmann distribution [1]:

$$P(E_i) = \frac{1}{Z}\exp(-E_i/T) \tag{3.1}$$

where the partition function $Z = \sum_j \exp(-E_j/T)$. In attempting to carry this mechanism over to a population-oriented paradigm such as genetic algorithms, we focus on generating a Boltzmann distribution across a population and time.

To do this we make two assumptions:

1. Each individual in the population is in the neighborhood of all other distinguishable individuals.

2. Differences in objective function value may be used to distinguish different classes of individual.

The first assumption is necessary to separate acceptance and neighborhood exploration. In the serial mechanism they are intertwined, an undesirable feature if one is interested in efficient parallel implementations or if one wants to perform more interesting exploration operators such as recombination, recoding, and the like. The second assumption is necessary to partition a population into equivalence classes. Of course, secondary criteria such as genotypic or phenotypic similarity may be used to distinguish individuals that have significant differences in inner or outer appearance even though their objective function values may be close.

With these assumptions, a skeleton population-oriented Boltzmann selection mechanism may be defined as follows:

1. Choose an individual uniformly at random (with or without replacement) from the current population.
2. Choose another individual with an objective function value different from the first by a threshold amount θ.
3. Half of the time, choose a third individual with an objective function value different from the first and second individuals by the threshold amount (this is called strict choice), and the other half of the time, choose the third individual with an objective function value different by the threshold amount from the first individual alone (this is called relaxed choice).
4. Hold a secondary (*anti-acceptance*) competition between individuals two and three, keeping individual two with the anti-acceptance probability $p' = \exp(-E_3/T)/[\exp(-E_2/T) + \exp(-E_3/T)]$.
5. Hold the primary (acceptance) competition between the winner of the anti-acceptance competition and individual one, keeping individual one with the usual logistic acceptance probability $p = \exp(-E_1/T)/[\exp(-E_1/T) + \exp(-E_{\text{win}}/T)]$, where E_{win} is the function value of the winner of the anti-acceptance competition.
6. Repeat at step 1 until the new population is full.

The steps are straightforward, but why do we need to choose competitors to have different function values, and what is the purpose of the anti-acceptance competition?

In serial simulated annealing, there is never any danger of comparing two instances of the same individual during the acceptance step, because a neighbor is always generated. In a populations approach, we imagine the other operators acting (or not acting) asynchronously and must guard against comparing an individual to a copy of itself or another member of its equivalence class. If we do not do this, the procedure will not be able to maintain the Boltzmann distribution stably and will ultimately fill the population with copies of the best individual in a manner similar to other GA selection schemes.

The anti-acceptance competition seems counterintuitive until we recognize that we are simply trying to create the population equivalent of generating a neighbor uniformly at random. Intuitively this may be understood by recognizing that if the algorithm is successful, it will create a steady proportion of individuals according to the Boltzmann distribution. If a uniform random sample is drawn from a Boltzmann-distributed population, it will be biased toward better individuals. To counteract this tendency, we simply choose a competitor for the first individual by picking two individuals from the population, thereafter choosing the winner of this secondary competition

using the complement of the usual acceptance probability. Since the complementary probability prefers poorer individuals, the distribution following anti-acceptance will be considerably flattened with respect to the original Boltzmann distribution.

To see this a bit more formally, we consider the effect of performing anti-acceptance on a Boltzmann-distributed set of alternatives. We start by considering the case of relaxed anti-acceptance, analyzing its performance at both high and low temperatures. Thereafter, the performance of strict anti-acceptance is considered, and the need for a 50-50 split between strict and relaxed anti-acceptance is demonstrated.

3.1 Analysis of anti-acceptance

To analyze the flattening effect of anti-acceptance, we must calculate the transition probabilities of the anti-acceptance matrix, a computation that proceeds analogously to the computation of the transition probabilities for normal simulated annealing [1].

In relaxed anti-acceptance the off-diagonal terms of the transition matrix may be calculated as follows:

$$P'_{ij} = \frac{\exp(-E_j/T)\exp(-E_i/T)}{Z[\exp(-E_i/T) + \exp(-E_j/T)]} \tag{3.2}$$

This is so because the transition occurs when the transition is accepted (as given by the complement of the usual logistic acceptance probability) and when the jth individual is chosen from the Boltzmann-distributed population.

The on-diagonal terms are obtained by taking a row sum of off-diagonal terms and subtracting the total from one:

$$P'_{ij} = 1 - \sum_{i \neq j} P'_{ji} \tag{3.3}$$

Assuming that the system is at its desired fixed point — assuming the p_i vector is Boltzmann — we calculate the probabilities p'_j after anti-selection as follows:

$$p'_j = \sum_i p_i P'_{ij} \tag{3.4}$$

Recognizing that $P'_{ij} = P'_{ji}$ for the case of relaxed anti-acceptance, after some algebraic manipulation, we obtain the following equation:

$$p'_j = p_j + \sum_{i \neq j} (p_i - p_j) P'_{ij} \tag{3.5}$$

Some further substitution yields the equation

$$p'_j = p_j + \frac{\exp(-E_j/T)}{Z} \sum_{i \neq j} \frac{\exp(-E_i/T)}{Z} \left[\frac{\exp(-E_i/T) - \exp(-E_j/T)}{\exp(-E_i/T) + \exp(-E_j/T)} \right] \tag{3.6}$$

At very high temperatures, clearly $p'_j = p_j$, and this yields uniform neighborhood visitation as desired, because a Boltzmann distribution at high temperature is flat. A more careful analysis of the change in the distribution at high temperatures may be performed by recognizing that for small x, $\exp(-x) \approx 1 - x$. Substituting into the previous equation yields the following result:

$$\Delta p_j = p'_j - p_j = p_j \frac{E_j - \bar{E}}{2T} \tag{3.7}$$

Thus, anti-acceptance flattens the distribution at high temperatures by an amount related to the difference between a particular function value and the average. The ideal change in distribution may be calculated, recognizing that we would like to maintain a uniform neighborhood:

$$\Delta p_j = \frac{1}{n} - p_j = p_j \frac{E_j - \bar{E}}{(1 - E_j/T)T} \tag{3.8}$$

where n is the number of separate equivalence classes. Thus, at high temperatures the anti-acceptance procedure flattens the distribution with proper functional form, albeit with some disagreement over the proper constant of proportionality. Nonetheless, the flattening mechanism approximates a uniform neighborhood over those individuals that maintain significant Boltzmann proportions. An analysis at low temperatures helps complete the picture of relaxed anti-acceptance performance.

At low temperatures, the best function value E^* dominates. It may be shown easily that the distribution in the limit is Boltzmann ($\Delta p_j \to 0$), thereby allocating all anti-acceptance trials to the best class. Clearly this is undesirable as it removes all notion of keeping the competition alive uniformly in the neighborhood. As a result, it is important to add strict anti-acceptance to maintain the desired behavior at low temperatures.

The analysis of strict anti-acceptance is similar to that performed for the relaxed form. Calculating the transition probabilities must account for the selection of a competitor without replacement, however. Under this assumption, the following equation is obtained:

$$P'_{ij} = \frac{\exp(-E_j/T)\exp(-E_i/T)}{Z_i[\exp(-E_i/T) + \exp(-E_j/T)]} \tag{3.9}$$

where $Z_i = Z - \exp(-E_i/T)$. Recognizing that $P'_{ji} = (Z_i/Z_j)P'_{ij}$ we obtain the following result for strict anti-acceptance:

$$\Delta p_j = \frac{\exp(-E_j/T)}{Z} \sum_{i \neq j} \frac{\exp(-E_i/T)}{Z_i} \left[\frac{\exp(-E_i/T) - \frac{Z_i}{Z_j}\exp(-E_j/T)}{\exp(-E_i/T) + \exp(-E_j/T)} \right] \tag{3.10}$$

At high temperatures, strict and relaxed anti-acceptance have virtually identical performance, because $Z_i \approx Z_j \approx Z$. At low temperatures, an interesting difference is observed. Simple asymptotic analysis shows that all of the strict anti-acceptance trials are given to the second-best individual by strict anti-acceptance, a reasonable result if we recall that strict choice requires selection of two different individuals during the anti-acceptance step. By itself strict anti-acceptance is also undesirable. We would really like to have a uniform choice between the last remaining competitors (the best and the second best). The method selected to accomplish this is to choose between strict and relaxed anti-acceptance randomly, using each fifty percent of the time. Thus, at low temperature, the procedure approaches a uniform distribution over the last two remaining competitor classes of any significance. At intermediate temperatures, the average of strict and relaxed anti-acceptance gives a moving almost-uniform selection among remaining significant competitors. Since the conditions of a uniform selection in the neighborhood of an individual are approximately satisfied, we therefore expect the steady distribution of the mechanism to be roughly Boltzmann by standard theorems of simulated annealing [1].

With this theoretical background we turn to implementing and testing Boltzmann tournament selection.

4. Implementation and proof-of-principle testing

A Pascal version of Boltzmann tournament selection is described, and preliminary test results are presented over a range of temperatures.

4.1 An implementation in Pascal

A version of Boltzmann tournament selection has been implemented in Turbo Pascal 5.5. Global declarations specific to the selection routines are shown in figure 1. Utility code is shown in figure 2, including the `logistic` function, the `fthreshold` function for calculating an objective function threshold from probability gap and temperature, and procedure `makeshuffle` for generating a random permutation of specified length. The procedure `preselect` creates another random permutation and initializes the `pick` pointer to 1.

The function `chooseother` is used to pick individuals from the population that have function values that are different from the values `f1`, and `f2` by an amount of `threshold` or more. If the selection process is unsuccessful, the last individual chosen is used.

```
{ boltzsel.pas: Boltzmann tournament selection }
{ 4-24-90  David E. Goldberg                   }

{ declarations }
type  shufflearray = array[1..maxpop] of integer;

var   pick:integer;                  { global tournament select vars }
      round:shufflearray;
```

Figure 1: The global declarations for Boltzmann tournament selection are few. Other global declarations and the overall style of implementation are similar to the code of Goldberg [7].

The function `select` implements Boltzmann tournament selection with weighted strict-relaxed anti-acceptance as described in section 3. The function `select` may be called to choose individuals for subsequent genetic processing in the normal manner [7]. Note, in the function `chooseother` that only a proportion check of the population may be checked to obtain an individual who is different from one or the other of the individuals already chosen by an amount greater than `threshold`. The probability of succeeding in this effort reduces as the temperature reduces and the population becomes dominated by one class of individual; this is not a problem, however, because as the temperature reduces, the performance of Boltzmann tournament selection should approach that of deterministic tournament selection, and the lack of different competitors encourages this transition. A more efficient implementation might hard wire deterministic selection at some specified temperature to avoid unnecessary difference checking in `chooseother`.

In practical implementations where the effective boundaries between different classes of individual are likely to change, a variable thresholding mechanism is probably desirable. The function `fthreshold` calculates a threshold in function value difference corresponding to a fixed gap between the logistic acceptance probabilities between two alternatives, `gap` $= p_{\text{better}} - p_{\text{worse}}$. Thus, for fixed gap, the threshold dividing classes will reduce as the temperature is reduced. No doubt, other reasonable thresholding mechanisms can be devised.

4.2 Preliminary testing

Some preliminary simulations are performed using this code. Five classes are defined with function values for the jth class given as $E(j) = j$.[2] A fixed threshold of 0.5 is used and each simulation is run for a hundred generations with a fixed `popsize` = 200. Each class is initialized with 20% of

[2]The symbol `f` is used in the code instead of E to conform with the practice in Goldberg [7]. The code is written assuming that `f` is to be minimized.

```
function logistic(x:real):real;
const limit = 11.513;
begin
 if x>limit then logistic := 1.0
  else if x < -limit then logistic := 0.0
   else logistic := 1.0/(1.0+exp(-x))
end;

function fthreshold(gap,temperature:real):real;
{ calculate threshold for distinguishing classes from f values }
const default = 1.0e10;
var prob:real;
begin
  if gap <= 0 then gap := abs(gap);
  prob := 0.5 + gap*0.5;
  if prob >= 1.0 then fthreshold := default
   else fthreshold := -temperature * ln(1/prob-1);
end;

procedure makeshuffle(n:integer; var shuffle:shufflearray);
{ makes an n-permutation }
var j, other, temp:integer;
begin
{initialize}
 for j:=1 to n do shuffle[j] := j;
{shuffle}
 for j:=1 to n-1 do begin
  other := rnd(j,n);
  temp := shuffle[other];
  shuffle[other] := shuffle[j];
  shuffle[j] := temp
 end;
end;
```

Figure 2: The utility code required for Boltzmann tournament selection includes `logistic`, `fthreshold`, and `makeshuffle`. Additionally, a random number generator such as that contained in Goldberg [7] is required with a function `rnd(lo,hi)` that returns a uniformly distributed random integer on the specified interval `[lo, hi]`. A function `flip(probability)` that returns `true` with specified `probability` is also required.

```
procedure preselect(var pick:integer;
                        popsize:integer;
                    var round:shufflearray);
begin {set pick counter and tournament round}
 pick := 1;
 makeshuffle(popsize, round);
end;
```

Figure 3: The procedure **preselect** initializes the random permutation **round** and the counter **pick**, which are used for doing random selection of individuals from the population without replacement.

```
function chooseother(popsize:integer; var pop:population;
                     f1, f2, threshold:real):integer;
{ choose an individual with some other f value }
const check = 0.1; { check a tenth of the population before quitting }
var   other, count, checksize:integer;
begin
 count := 1;
 checksize := trunc(check*popsize);
 repeat
  other := rnd(1,popsize);
  count := count + 1;
 until ((abs(f1-oldpop[other].f) > threshold)
        and (abs(f2-oldpop[other].f) > threshold))
        or  (count > checksize);
 { quit when different by threshold or count too big }
 chooseother := other;
end;
```

Figure 4: The function **chooseother** is used to choose individuals who are different in objective function value from previously chosen individuals.

the population's individuals. Table 2 shows the results, comparing the average proportion of trials in each class over 100 generations to the proportion given by a Boltzmann distribution at four temperatures T = 100, 10, 1, 0.1. The simulations track the Boltzmann distribution quite closely. Since no neighborhood generation was used in these runs, once a class is absorbed it is gone forever, but only classes with expected low proportions were absorbed in the simulations. The restoring pressure to keep the population Boltzmann is quite strong, although users familiar with the rock stability of sharing functions [4, 5, 11] will observe that BMTS bounces around a bit more. Nonetheless, the potential for parallelizing this procedure efficiently

outweighs its somewhat greater variance. We briefly consider parallelization of the routine and some of its other uses and extensions in the next section.

5. Uses and extensions

Boltzmann tournament selection was developed primarily as a means to carry over Boltzmann distributions and cooling schedules to genetic algorithms, thereby guaranteeing asymptotic convergence to globally optimal structures in GAs as well. This is a worthy goal in its own right, but in so doing a mechanism has been created that may be used and extended in a number of ways:

1. It may be implemented efficiently on parallel hardware.
2. It may be used as a function-value-based *niching* mechanism.
3. Other forms of Boltzmann selection may be implemented under proportionate selection schemes using *sharing functions*.

It is interesting that in many efforts to parallelize simulated annealing, researchers have concentrated on more or less accurately implementing the one-at-a-time version of SA on parallel hardware. This has ranged from the so-called error algorithm that runs noncommunicating serial algorithms in parallel (and is called the error algorithm, because of the "errors" that occur when individual processors do not use the current best structure in the Metropolis acceptance step) to more cooperative versions that communicate and try to collectively achieve the temporal Boltzmann distribution. It is surprising that more effort has not been directed at achieving the distribution across a population. It is, after all, the distribution that is important to ultimate convergence and not any particular method of attaining the distribution. Thus, Boltzmann tournament selection creates the potential for fast parallel implementations of both GAs and SAs. Imagine an implementation with one processor per individual. Since pairwise comparisons are all that are required, Boltzmann tournament selection can be easily realized in parallel. This is not unlike efforts to parallelize other selection mechanisms in genetic algorithms [12, 15, 16, 18]. With only localized communication, pairwise exchanges of individuals between randomly chosen processors can be used to keep a state of panmixia to prevent biased sampling in any particular geographic neighborhood.

We note that the distribution mechanism advocated here may be viewed as a means to achieve *niching* based on difference in objective function values. Elsewhere, various mechanisms have been advocated for encouraging the stable formation of clusters of individuals [3–5,11]. These mechanisms stably allocate more copies to better individuals and fewer copies to weaker individuals. Recalling that the mechanism developed in this paper attains proportions in the population that are roughly Boltzmann, $p_j = \exp(-E_j/T)/Z$. Note that such a distribution is niching-like in that it allocates more copies to better classes and fewer to worse. It is interesting that the convergence

```
function select(popsize:integer;
                var pop:population;
                var pick:integer;
                var round:shufflearray):integer;
{ Boltzmann tournament selection }
const bias = 0.5; { randomization for third chooseother }
var first, second, third:integer;
    probf, deltaf:real;
begin
 if pick > popsize then preselect(pick,popsize,round);
 { hold a competition }
 first  := round[pick];
 { anti-acceptance probability to flatten distribution }
 second := chooseother(popsize,pop,pop[first].f,pop[first].f, threshold);
 { randomization for strict vs. relaxed anti-acceptance }
 if flip(bias) then
    third := chooseother(popsize,pop,pop[first].f,pop[second].f,threshold)
  else third := chooseother(popsize,pop,pop[first].f,pop[first].f,threshold);
 probf  := logistic((pop[third].f - pop[second].f)/temperature);
 if flip(probf) then second := third;
 { primary selection according to Boltzmann acceptance criterion }
 deltaf := pop[second].f - pop[first].f;
 probf  := logistic(deltaf/temperature);
 if flip(probf) then select := first else select := second;
 { increment pick }
 pick := pick + 1;
end;
```

Figure 5: The function `select` implements Boltzmann tournament selection.

T	BM/BMTS	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
1.00	BM	0.2040	0.2020	0.2000	0.1980	0.1960
	BMTS	0.2078	0.2044	0.2101	0.1891	0.1887
10.0	BM	0.2419	0.2188	0.1980	0.1792	0.1621
	BMTS	0.2780	0.2355	0.2113	0.1631	0.1122
1.0	BM	0.6364	0.2341	0.0861	0.0317	0.0117
	BMTS	0.6604	0.2513	0.0832	0.0041	0.0011
0.1	BM	1.0000	0.0000	0.0000	0.0000	0.0000
	BMTS	0.9755	0.0176	0.0052	0.0018	0.0000

Table 2: Comparison of Boltzmann tournament selection (BMTS) versus a Boltzmann distribution (BM) at four temperature values.

guarantees of simulated annealing derive almost exclusively from this use of what might be termed stepwise temporal niching. This seems consonant with the argument made elsewhere [7, 11] that diversity should rarely be maintained or introduced for its own sake, but instead previously useful diversity should be "recalled" spatially through niching; however, we then recall that whether or not a serial SA can recall an individual is entirely dependent upon whether that individual is sufficiently close to the current individual and whether intermediate function value barriers have not risen so much as to prevent its access. This is certainly one of the unsung difficulties of serial simulated annealing, one that can be overcome by taking the populations view adopted herein.[3]

It is also straightforward to ask whether niching can be performed using Boltzmann tournament selection in such a way as to allow separate stable clusters to form around individuals who have similar function values, but very different outer or inner appearance. This may be accomplished quite simply by using a secondary criterion such as phenotypic or genotypic distance to distinguish between individuals who have similar function values, but who are in distinct niches. To implement this mechanism, appropriate logic can be added to the `chooseother` routine to require difference on the basis of objective function value or difference in genotype or phenotype.

Such associations also suggest a method for implementing Boltzmann selection in genetic algorithms that use proportionate selection schemes. The sharing function method described elsewhere [4, 5, 11] may be adapted directly if $f = \exp(-E/T)$ is used as a fitness function, and distance is measured as the absolute value of the difference in f between pairs of strings. It is interesting that Boltzmann tournament selection and this sharing function approach generate similar outcomes even though their mechanisms are very different (the mechanisms are almost dual in nature). Sharing works by forcing groups of like individuals to share some limited resource. Boltzmann tournaments work by forcing individuals to compete outside their niche. The restoring mechanism of this latter mechanism is inobvious until we realize that this forced competition of unlike individuals causes currently out-of-favor individuals to be in many more competitions than they would if they were selected uniformly at random from the existing distribution.

6. Conclusions

In this note, the method of Boltzmann tournament selection has been described, implemented, and partially tested. The method carries the ideas of a Boltzmann distribution and thermal equilibrium to genetic algorithm

[3] A more remote connection with niching and trial allocation has been made in a paper that argues for a mixed strategy when confronting a mix of probabilistic and deterministic decision problems [6]. There the view is closer to the temporal orientation of serial simulated annealing, but the temporal view is justified by the low cardinality of the decision set. It is possible that the mechanisms advocated here might be useful in simplifying the structure of the bidding mechanism suggested for achieving the desired mix between decisive and fuzzy decision-making.

practice, thereby guaranteeing the asymptotic convergence in genetic algorithms that has long been enjoyed by simulated annealing. The technique may be parallelized easily, permitting efficient implementations of simulated annealing algorithms and Boltzmann genetic algorithms on many types of parallel machine. Moreover, this mechanism has been viewed as a form of niching, and connections to other niching mechanisms have been drawn.

The mechanism is ready for testing in practical problems and for implementation on parallel hardware. Perhaps more importantly, the existence of this mechanism may serve to bridge simulated annealing and genetic algorithms to the ultimate benefit of both.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant CTS-S451610. Support was also provided by the Alabama Research Institute.

References

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing* (Wiley, Chichester, 1989).

[2] L. Davis and M. Steenstrup, "Genetic algorithms and simulated annealing: An overview." In *Genetic Algorithms and Simulated Annealing*, L. Davis, ed. (Pitman, London, 1989) 1–11.

[3] K.A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, doctoral dissertation, University of Michigan. *Dissertation Abstracts International*, **36(10)** (1975) 5140B. (University Microfilms No. 76-9381.)

[4] K. Deb, *Genetic algorithms in multimodal function optimization*, Master's thesis and TCGA Report No. 89002 (The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa, 1989).

[5] K. Deb and D.E. Goldberg, "An investigation of niche and species formation in genetic function optimization," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 42–50.

[6] D.E. Goldberg, *Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding*, TCGA Report No. 88002 (The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa, 1988).

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

[8] D.E. Goldberg, "Genetic algorithms and Walsh functions: Part I, A gentle introduction," *Complex Systems*, **3** (1989) 129–152.

[9] D.E. Goldberg, "Genetic algorithms and Walsh functions: Part II, Deception and its analysis," *Complex Systems*, **3** (1989) 153–171.

[10] D.E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systems* (1989) 493–530.

[11] D.E. Goldberg and J.J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, (1987) 41–49.

[12] M. Gorges-Schleuter, "ASPARAGOS: An asynchronous parallel genetic optimization strategy," *Proceedings of the Third International Conference on Genetic Algorithms*, (1899) 422–427.

[13] J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, MI, 1975).

[14] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science*, **220(4598)** (1983) 671–680.

[15] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 428–433.

[16] H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," *Proceedings of the Third International Conference on Genetic Algorithms*, (1989) 416–421.

[17] D.J. Sirag and P.T. Weisser, "Toward a unified thermodynamic genetic operator," *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, (1987) 116–122.

[18] J.Y. Suh and D. Van Gucht, *Distributed Genetic Algorithms*, Technical Report No. 225, Computer Science Department, Indiana University, 1987.