Distributed Evolutionary Algorithm Topologies with Adaptive Migration Schemes

Muhammad Hijaze  
School of MACS  
Heriot-Watt University  
Edinburgh, UK  
mh132@hw.ac.uk

David Corne  
School of MACS  
Heriot-Watt University  
Edinburgh, UK  
d.w.corne@hw.ac.uk

Abstract—Distributed evolutionary algorithms are of increasing interest and importance for three main reasons: (i) a well designed dEA can outperform a ‘standard’ EA in terms of reliability, solution quality, and speed; (ii) they can (of course) be implemented on parallel hardware, and hence combine efficient utilization of parallel resources with very fast and reliable optimization; (iii) parallel hardware resources are increasingly common. A dEA operates as separate evolving populations with occasional interaction between them via ‘migration’. A specific dEA is characterized by the topology and nature of these interactions. The performance of alternative topologies and migration mechanisms in this field remains under-explored. In this paper we continue an investigation of two simple, novel dEA topologies, comparing with a cubed-based topology that underpins Alba et al’s GD-RCGA (a state of the art dEA). The focus in this paper is on testing a novel adaptive migration scheme, in which the frequency of migration events adapts dynamically in response to the current balance between exploration and exploitation. We also focus on high dimensional versions of a selection of hard function optimization problems. We find that the adaptive migration scheme is promising, and that overall results marginally favour a simple three-level tree-based topology and adaptive migration with a longer window, especially as dimensionality increases.

Keywords-component; distributed evolutionary algorithm; function optimization

I. INTRODUCTION

Distributed evolutionary algorithms (dEAs) operate by having several independent populations of chromosomes (candidate solutions), with occasional interaction between them – this interaction usually takes the form of selected chromosomes moving (or being copied) between populations. In a typical dEA, the separate populations will evolve independently for some amount of time, or a fixed number of generations, and then interaction in the form of migration will happen. During a migration event, chromosomes from one or more of the populations (sometimes called ‘demes’) will be copied into one or more of the other populations. The populations then continue to evolve independently until the next migration event, and so it continues. This style of distributed EA design is known to have several good performance properties. For example, dEAs are well known to be capable of performing more successfully than standard EAs that are otherwise the same’ (e.g. in terms of the choices of genetic operators and total population size) – that is, when implemented on a single processor, both the solution quality and speed (in terms of total fitness evaluations) can be very favorable in comparison to non-distributed EA designs that have a single (so-called panmictic) population [1—5]. Secondly, dEAs are, naturally, highly parallelizable, and therefore offer a highly natural way to exploit a variety of different parallel hardware architectures. The latter advantage is becoming increasingly more important as parallel hardware resources become ever more common.

There are two main kinds of dEA (also generally called ‘parallel genetic algorithms’). In the simplest kind, a standard EA design is distributed over several processors in a straightforward manner, but otherwise is little changed algorithmically. The alterations to the EA design in such cases are essentially only those necessary to enable exploitation of the parallel hardware. Examples include [6,7]; e.g. the implementation is engineered so that, for example, many fitness evaluations can be done in parallel, but otherwise the flow of information and control is unchanged from the serial case.

The second (and main) line of research in distributed EAs involves algorithm designs that establish independent populations, often called subpopulations or demes, each (for example) using its own processor. In such a scheme, a ‘migration’ strategy is used to communicate information between processors at intervals [8]. There is an immense space of potential alternatives for migration schemes. The common migration scheme is typically to send copies of good chromosomes from some populations directly to other populations (where they may for example, overwrite the worst chromosomes in the receiving subpopulations). When the broad dynamics of such a dEA are considered, it is intuitively clear that this form of migration corresponds to balancing useful exploration in the search (since there are independent subpopulations, mostly non-interacting) with occasional exploitation (promoted by migration), in a way that is not achieved by standard single-population EA designs. As mentioned, this typically leads to improved performance in terms of both solution quality and speed.
Distributed EAs are usually described as either fine-grained or coarse-grained, depending on the sizes of the sub-populations. The extreme version of a fine-grained model is where each chromosome is evaluated on a single dedicated processor (e.g. [9]). A more favored approach, which is more justified in terms of prevailing hardware configurations, is the coarse grained model (e.g. [3,9—14]. The Genitor algorithm [3] is a particularly well-known example, in which the subpopulations are linked in a ring topology. During a migration event, chromosomes are distributed to immediately neighbouring nodes on the ring. A more recent example is that of [12], in which the populations are linked in a cube-based topology, in which subpopulations are at vertices of the cube, and each is linked to three others along the edges of the cube.

We explore here a topology that carries the main design characteristics of the cube topology in [12], continuing an investigation begun in [15]. However the dEA in [12] has several other sophistications that we do not implement, since we are currently interested in understanding the performance of alternative simple topologies. In [15], we investigated three simple parallel EA topologies. The first, T1, was a collection of 8 (or 16) subpopulations, each of which communicated to the master node whenever it achieved a new best chromosome fitness. At intervals, the master would deliver a copy of the current global best to the subpopulation currently with the worst progress. In T2, the 8 (16) subpopulations were instead 4 (8) subgroups of 2 subpopulations. Populations in a subgroup of 2 regularly communicate their bests to each other, and the master node would regularly deliver a copy of the global best to the ‘worst’ subgroup. T3, as indicated, was modeled on Alba et al’s GD-RCGA [12], with a slightly different (but still ‘fixed-parameter’) migration scheme. Fuller details of T3 are provided later. In [12], we investigated these topologies on 30-dimensional versions of certain of the function optimization problems from the CEC2005 competition, and found that all performed very well (commonly solving these problems to optimality within the maximum time allowed), but T2 was marginally more successful in terms of success-rate and speed.

We speculate that the differences in performance of the three topologies lay in the different balances in each between exploration and exploitation. This in turn suggests that online adaptation of this balance may be beneficial. In this paper we therefore explore a simple extension to the basic migration scheme in which the probability of a migration event (and therefore the frequency of migration) changes according to the current progress of the distributed search. Progress is simply measured by the frequency with which the master node is receiving updates to the global best solution – when this is high, we assume that exploratory activity (ongoing search in the independent subpopulations) is going well, and we can reduce the probability of migration. Vice versa the probability of migration increases when updates in the global best become less frequent. In this paper we study again the three dEA topologies explored in [15], but: (a) with the adaptive migration scheme, and (b) on higher dimensional versions of the function optimization problems (50D and 100D) (however at the time of writing the results for adaptive migration are not available for 100D problems; these will be available well in time if the paper is accepted). This enables us to assess the performance of the adaptive migration scheme, and also to assess whether our observations in [15] scale up to higher dimensional tasks.

The remainder of the paper is set out as follows. Section II describes our parallel EA control strategy and the basic migration mechanism that we use, and also the adaptive migration scheme. This section also describes topologies T1, T2 and T3. Section III then covers some experimental design details, including the test functions and the evaluation strategy. In section IV we present and discuss our results, and then we conclude in Section V.

II. ALGORITHM DESIGNS, ADAPTIVE MIGRATION SCHEME AND TOPOLOGIES

A. Master and Client Threads

Our dEAs are implemented in a physically parallel and asynchronous environment, using a collection of standard workstations via sockets technology. In all cases, the algorithm is controlled via a master thread and several client threads. The basic operation in all models, is as follows. Initially, connections are established, and the threads initialised; then, a master thread (M) receives continual updates from each client thread (C) whenever a client has achieved a new best fitness (locally, i.e. in its own subpopulation). When M determines that one of these chromosomes is a new global best, M sends this to the client that currently has the worst “best local fitness” according to M’s latest information. In the ‘fixed’ scheme, this happens with probability 1. However, when the migration scheme is adaptive, M distributes the best chromosome to the worst client with an adaptive probability, usually smaller than 1. Details are provided below. Meanwhile, client threads continue to operate the evolution of their own subpopulation on a single processor, and incorporate new chromosomes as and when they are received from M. Whenever a new chromosome is received by a client C, if it is fitter than the current best in that population, then it is included in the population and the current worst at C is discarded. At frequent intervals, each client sends its best chromosome to M.

The first model (T1) operates in precisely the way above, and uses a straightforward architecture in which the master is directly connected to each client. In models 2 and 3, the topology is altered, and the master thread connects to a restricted number of clients. In each case, the master connects to a group of clients, and the clients within a group communicate directly with each other. Below we show pseudocode clarifying the operation of the master and client threads. For T1, the pseudocode is exact. For T2 and T3, there are differences that will be clarified later. In essence, however, each of T2 and T3 defines a set of groups of clients, and each client operates as both a master and a client within its group, while an overall master thread operates over the groups.

The overall responsibilities of M are:

1. Establish connections between the clients
2. Establish and initialize parameters
3. Start and Terminate the optimization
4. Receive and store up to date data from clients
5. Distribute appropriate data to clients

The master pseudo code operates three threads, as follows:

**Master Thread 1:**
Connect Master with Clients;
For each Client:
    Send all parameters;
    Send “Start Process” token;
End for
Run Thread2; // for receiving data
Run Thread3; // for sending data

**Master Thread2:**
Repeat
    For each Client
        Listen to Connection Stream and store received data in DataString;
        Decode DataString and Convert it to real Chromosome and Fitness;
        If OptimChrom = Null // first one
            Set OptimChrom;
            Set OptimFitVal;
        Else if this Fitness is better than OptimFitVal
            Update OptimChrom;
            Update OptimFitVal;
        End if;
    End for;
Until target Fitness Value reached, or max time reached

**Master Thread3:**
Repeat
    If there is a new OptimChrom
        With probability $p$ Send OptimChrom and its fitness to worst client;
        Sleep(100);
    Until Reach target Fitness Value;

The client threads operate as follows. Thread 1 connects to and receives data from M. It connects with the master, and waits for a “Start Process” signal. When this is received, it starts threads 2 and 3. Thread 2 runs the evolutionary algorithm on that client’s processor, and thread 3 deals with sending updates of this client’s best chromosome to the master.

**Client Thread 1:**
wait for “Start Process” signal from master;
Run Thread 2;
Run Thread 3;
While connection is established
    Store received data in DataString;
    Decode and Convert DataString to 
        NewChrom and its FitVal;
    If new FitVal is better than 
        Current OptimFitVal
        Update OptimChrom; //NewChrom
        Update OptimFitVal; // New FitVal
    Replace worst chromosome in 
        Population with NewChrom
End if
End While

**Client Thread 2:**
Create and Initialize Population;
Set Count = 0;
Set SendingNewDataCount=0;
Find best chromosome and call it OptimChrom;
Set OptFitVal;
Repeat
    Count++;
    Select chromosomes c1 and c2
    Crossover(c1,c2), producing children c1’ and c2’
    Mutate(c1’), producing c1’’
    Mutate(c2’), producing c2’’
    Calculate fitness of c1’’;
    Calculate fitness of c2’’;
    If fitness(c1’’) better than OptFitVal
        Set OptFitVal = fitness(c1’’)
        Set OptChrom = c1’’
        SendingNewDataCount++
        If SendingNewDataCount % 25 == 0
            SendNewData=True;
            If fitness(c2’’) better than OptFitVal
                Set OptFitVal = fitness(c1’’)
                Set OptChrom = c1’’
                SendingNewDataCount++
                If SendingNewDataCount % 25 == 0
                    SendNewData=True;
        Until OptFitVal <= TargetFitVal or
            Count=MaxCount
Until OptFitVal <= TargetFitVal or
    Count=MaxCount

**Client Thread 3:**
While Thread2 is running
    If SendNewData
        Send OptChrom and its fitness 
            to Master;
        SendNewData=False;
        Sleep(100);
    End If;
End While;
B. Evolutionary Algorithm and Implementation Details

All topologies use a generational evolutionary algorithm that operates with a truncation selection strategy [14], in which, in each generation, the best 33% of the population are retained, and, selecting only from this best 33%, crossover and mutation are employed to generate the remaining 66% of the new population. We use the crossover and mutation operators used by Alba et al. in their dEA GD-RCGA, detailed in [12]; these are the fuzzy connective-based crossover operators: F-Crossover, S-Crossover, L-Crossover, M-Crossover, along with one-point, two-point, and uniform crossover. Each time a crossover operator is applied (crossover probability is 0.6), a choice between these seven operators is made uniformly at random. Mutation (Gaussian mutation of a single randomly chosen parameter) is performed on a child of crossover with probability 0.25.

All our models use 15 individuals per subpopulation and, in the fixed scheme, a migration is performed every 25 generations. There is a predefined maximum number of generations (10000), but a trial run will terminate if it has reached the target fitness values. The physical hardware used is a cluster of eight personal computers running Microsoft widows XP Professional SP3, each one having an Intel Pentium IV 2.99 GHz processor and 2 GB of memory. The machines are interconnected by a Fast-Ethernet (100 Mbps) network.

In the adaptive migration schemes, all is the same as above except that, when a migration event is due, it happens with a probability \( p \). The master thread keeps track of the latest (local) best-chromosome updates from each client, and knows how these updates are distributed across clients. That is, for the last \( w \) updates (here we test \( w=300 \) and \( w=100 \)), the client knows the proportion \( u_i \) of those updates that are due to each client \( i \). When it is time to send the global best to the worst client (the client \( x \) whose local best is the least fit — breaking ties randomly), this happens with probability \( 1-u_x \). So, if the worst client has nevertheless been improving its best fitness quite frequently, there is more chance that it will be left alone to continue to explore. However if the client’s population is relatively stagnant, there is more chance that it will be reinvigorated with the current global best chromosome.

C. Topologies

In topology 1 (T1), the master connects directly to each of 16 clients, and clients operate two per processor (i.e. are distributed over 8 machines). This is illustrated in Figure 1.

Toplogy 2 (T2) differs in a straightforward way from T1. As illustrated in Figure 2, the master in T2 connects directly with each of eight groups, and each group consists of two communicating clients. Recall that, with T1, the master keeps track of the current global optimum, and will copy this (with a probability) to the current worst client whenever a new when a migration is due. This happens in T2 at the group level; so, in T2, a new global best chromosome will first be transferred to the sister client in its group, and will soon appear in both clients of the previously worst group.

![Figure 1. Topology T1; the master connects directly with each of 16 clients.](image1)

![Figure 2. Topology T2; the clients are organised into groups of 2, and the master connects with each group.](image2)
this every 25 generations. At the master level, there are just
two groups; every 25 generations, the best chromosome within
the first group (i.e. from all populations in the first cube) is
sent to a randomly chosen client from the second cube. The
same then happens vice versa.

We tested both 8-population and 16-population examples
of each model in [15]; here we show results for 16-
subpopulation versions only, which tie up with Figures 1, 2
and 3 as shown. (In the case of T3, the 8-subpopulation model
comprised a single cube and hence did not involve migration
at a higher level than the cube itself.)

![Figure 3. Topology T3 – two connected cubes (each cube is based on Alba
et al’s model [12].)

III. EXPERIMENTS

A. Test Functions

We compare T1, T2 and T3 by using six of the well-known test
functions from the CEC2005 competition [16], and use two
versions of each – the standard version, and the ‘shifted’
version in which the global optimum is subject to random
translation, rendering the function less decomposable.
Specifically we work with the Sphere, Rosenbrock, Schwefel,
Rastrigin, Griewangk and Ackley functions, in 30, 50 and 100
dimensional, and shifted and non-shifted versions. Selection of
functions was simply pragmatic in terms of time and
processing constraints.

All results shown here reflect 20 independent trials on each
of these 12 test functions. We show either success rate, which
is the number of trials, out of 20, in which the function was
solved to optimality, or we show mean execution time, which
in all cases is measured over successful runs only. Note that
there is no particular prior expectation that success rate will be
large (or even above 0) in many cases, however the dynamics
of parallel search, in conjunction with the operators chosen
from [12], can be highly effective and we see that success rates
are often high for quite difficult problems in our parallel
asynchronous implementations.

We first show the results for 30-Dimensional problems; in
this case we import results seen in [15] for the ‘fixed’ case, and
compare with the adaptive migration strategy explored in this
paper.

<table>
<thead>
<tr>
<th>Function</th>
<th>1-client</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>18 (20)</td>
<td>19 (18)</td>
<td>18 (20)</td>
<td>18 (20)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>19 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Rosenbrock</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>10 (14)</td>
<td>18 (20)</td>
<td>18 (20)</td>
<td>18 (18)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>19 (15)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>18 (16)</td>
<td>17 (20)</td>
<td>15 (20)</td>
<td></td>
</tr>
<tr>
<td>Rastrigin</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (12)</td>
<td>20 (18)</td>
<td>20 (17)</td>
<td>20 (18)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Schwefel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>14 (10)</td>
<td>11 (17)</td>
<td>18 (17)</td>
<td>11 (17)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>19 (17)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (18)</td>
<td>19 (18)</td>
<td>20 (18)</td>
<td></td>
</tr>
<tr>
<td>Griewangk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (14)</td>
<td>18 (18)</td>
<td>20 (20)</td>
<td>18 (17)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (18)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Ackley</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (16)</td>
<td>20 (19)</td>
<td>20 (20)</td>
<td>18 (18)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td></td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (18)</td>
<td>20 (20)</td>
<td></td>
</tr>
</tbody>
</table>

Table I shows the success rates for both non-shifted and shifted
(in parenthesis) versions of the functions. For example, on the
Schwefel function, with the fixed (non-adaptive) scheme, T2
achieved 18 out of 20 successful runs on the shifted version
and 17 out of 20 on the non-shifted version. This table also
shows the results for a single-population serial EA that is
otherwise equivalent. The serial EA often performs well,
although this performance degraded significantly with
dimensionality and we do not show serial EA results later.

Since success rates tend to be quite high on the 30D
problems, relative performance of algorithms is hard to assess.
Even if we look at the problems that were clearly found more
difficult (especially by the serial EA), Rosenbrock and
Schwefel, again no very clear pattern emerges. However, if we
simply count the number of bold entries in each row (indicating
best in row, separately for shifted and non-shifted), ignoring
the single-population EA, we find that the fixed scheme
achieved the (equal or not) best success rate 22 times on these problems, compared 32 and 29 respectively for adaptive (300 and adaptive (100).

Turning now to mean execution times for successful runs, Figure 4 shows, from left to right, performance on each of the 6 problems in the order they appear in the Tables, considering only the Shifted versions – i.e. the first group of columns is Sphere, and the final group is Ackley.

In Figure 5, each cluster of three bars indicates performance of T1, T2 and T3 in order; these tend to show that T2 was overall best or close to best, especially considering Rosenbrock and Schwefel (the second and fourth group).

We now turn to the 50-dimensional results. First, Table II presents the success rates for 50D problems in a way analogous to Table I, but omitting the poor results for a serial EA.

**TABLE II. 50 D PROBLEMS SUCCESS RATES (OUT OF 20)**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (20)</td>
<td>20 (10)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>0 (20)</td>
<td>20 (6)</td>
<td>20 (18)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>0 (16)</td>
<td>0 (16)</td>
<td>0 (20)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>0 (11)</td>
<td>0 (18)</td>
<td>0 (18)</td>
</tr>
<tr>
<td>Rastrigin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (20)</td>
<td>14 (12)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (19)</td>
</tr>
<tr>
<td>Schwefel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (18)</td>
<td>18 (12)</td>
<td>20 (18)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (18)</td>
<td>20 (20)</td>
<td>20 (17)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>18 (15)</td>
<td>18 (18)</td>
<td>20 (15)</td>
</tr>
<tr>
<td>Griewank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (20)</td>
<td>16 (18)</td>
<td>16 (16)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Ackley</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td>20 (20)</td>
<td>12 (10)</td>
<td>16 (20)</td>
</tr>
<tr>
<td>Adap. (300)</td>
<td>20 (20)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
<tr>
<td>Adap. (100)</td>
<td>20 (19)</td>
<td>20 (20)</td>
<td>20 (20)</td>
</tr>
</tbody>
</table>
Table II again shows no obvious or striking pattern; again we see regular solution to optimality for these difficult functions, but also, moreso than we saw for 30D cases in Table I, frequent cases in which individual trials did not find the global optimum; this is especially striking in the Rosenbrock case. If anything, T2 seems marginally less successful overall than T1 and T2. Meanwhile, adaptive regimes seem better than the fixed regime: bold entries indicate (equal) best in a row (separately considered for shifted and non-shifted versions), and we get a rough feel for this by finding that the number of bold entries for the fixed regime amounts to 21, compared to 29 for adaptive (300) and 26 for adaptive (100).

**TABLE III. 100 D SUCCESS RATES (OUT OF 20)**

<table>
<thead>
<tr>
<th>Function</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>18 (14)</td>
<td>17 (20)</td>
<td>14 (16)</td>
</tr>
<tr>
<td>Rosenbrok</td>
<td>0 (19)</td>
<td>0 (20)</td>
<td>0 (14)</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>17 (18)</td>
<td>17 (19)</td>
<td>18 (12)</td>
</tr>
<tr>
<td>Schwefel</td>
<td>14 (14)</td>
<td>15 (16)</td>
<td>14 (14)</td>
</tr>
<tr>
<td>Griewangk</td>
<td>18 (16)</td>
<td>18 (18)</td>
<td>16 (14)</td>
</tr>
<tr>
<td>Ackley</td>
<td>16 (18)</td>
<td>16 (19)</td>
<td>14 (13)</td>
</tr>
</tbody>
</table>

Figure 7. Execution times (in ms) for 50D problem (Shifted versions only), in order Sphere, Rosenbrock, ..., Ackley. The three clusters in each group are, respectively, T1, T2 and T3, and within each of these the ordering is fixed, adaptive (300) and adaptive (100).

Figure 6 shows the relative execution times to finding the global optimum (in cases where it was found), oriented towards visualizing the relative performance of fixed, adaptive (300) and adaptive (100), analogously to Figure 4. Meanwhile, Figure 7 shows the same results oriented towards comparing T1, T2 and T3, analogous to Figure 5. The suggestion is that adaptive schemes are more reliably fast than the fixed scheme – spikes in Figure 6 tend to be associated with the fixed scheme (except in the case of the Schwefel function). Note that there is a gap in both of these plots in the place of the Rosenbrock function since this was never solved to optimality at 50D in the shifted version. Figure 7 also shows a general at least marginal advantage for T2 when an adaptive algorithm is used – noting that when 'slow' spikes occur in the T2 cluster, these are always associated with the fixed topology.

Finally, Table III shows results for 100 Dimensional problems with the fixed migration scheme (adaptive results not available at the time of writing). At 100 Dimensions, we start to see clearer differences in the performances of the topologies. We can see that T2 always achieved a unique best success rate in the case of the non-shifted functions. On the shifted functions, T1 and T2 achieved quite similar performance, both seeming to have an advantage over T3.

Figures 8 and 9 respectively show the performance in terms of execution time on the shifted (omitting Rosenbrock) and non-shifted versions. These show that T3 is fairly clearly the least favoured topology of the three, while, especially on the non-shifted functions, T2 seems to have a clear overall advantage.
IV. CONCLUDING DISCUSSION

We have noted that distributed evolutionary algorithms (dEAs) are of increasing importance for a variety of reasons. It is well known that parallelized optimization can provide more advantages than simply speed of execution; meanwhile the design of a distributed, asynchronous architecture leads to opportunities for managing exploration and exploitation in new ways, and the space of possible topologies and migration schemes has so far been little explored. We need to understand how to design dEAs to optimal effect, so more research is needed in this area.

In this paper our interest has been confined to some simple dEA topologies and migration schemes, of relevance to coarse-grained parallelization. E.g. each “processor” in our work is typically an subpopulation operating on a single processor, however it could easily be operating on (for example) a GPU cluster (perhaps operating its own fine-grained parallelism scheme), topologically isolated from other such clusters in the overall architecture.

We compared three simple topologies, one abstracted from Alba et al’s cube topology [12] and two simpler topologies, in the context of 12 function optimization problems, fixed or adaptive migration, and on 30, 50, and 100-dimensional versions of the problems (although at the time of writing, results for 100D problems are presented only for fixed migration.

Results were often inconclusive, largely because our dEAs each regularly performed with 100% success in finding the global optimum solutions, especially on 30D or 50D problems. However, we show evidence that adaptive migration schemes are beneficial in terms of both speed and success rate. However, generally, the adaptive schemes (especially that with the longer adaption window of the two trialed), were faster than the fixed scheme in reaching the optimal solution, and also more regularly achieved the higher success rates than did the fixed schemes. Evidence also suggested that T2 may have a slight advantage in speed and success rate on the less challenging landscapes, especially in the 100D case. We also saw in [15] that topology T2 was clearly and reliably the fastest topology for the 30D problems, with the fixed migration scheme (apparent here in Figures 4 and 5 but somewhat obscured by the quantity of information in those figures).

In conclusion, since it seems clear that coarse-grained dEA topologies are able to effect a healthy balance between exploration and exploitation, and this seems to be mediated by the migration scheme (or other schemes for interaction between subpopulations), it makes sense to explore dEA designs that have adaptive migration schemes. In this paper we explored a simple adaptive migration scheme, and generally found it was effective in terms of success rate and speed. Considering specifics of the topology of the dEA, we found that three simple topologies studied seemed to have different performance characteristics over the studied set of functions, but it was hard to discern a clear pattern. In future work we expect to consider a much larger collection of functions (in attempt to help clarify relative performance issues), and consider new variants of adaptive migration scheme.

REFERENCES