Improving Migration by Diversity

Jörg Denzinger
Department of Computer Science
University of Calgary
denzinge@cpsc.ucalgary.ca

Jordan Kidney
Department of Computer Science
University of Calgary
kidney@cpsc.ucalgary.ca

Abstract- We present an improvement to distributed GAs based on migration of individuals between several concurrently evolving populations. The idea behind our improvement is to not only use the fitness of an individual as criterion for selecting the individuals that migrate, but also to consider the diversity of individuals versus the currently best individual. We experimentally show that a distributed GA using a weighted sum of fitness and a diversity measure for selecting migrating individuals finds the known optimal solutions to benchmark problems from literature (that offer a lot of local optima) on average substantially faster than the distributed GA using only fitness for selection. In addition, the run times of several runs of the distributed GA to the same problem instance vary much less with our improvement than in the base case, thus resulting in a more stable behavior of a distributed GA of this type.

1 Introduction

Migration, i.e. the exchange (in the sense of copying) of individuals between otherwise divided populations, is the key concept of most distributed Genetic Algorithms that are based on multi-demes, that are also known as island model GAs (see [Gr85], [Ca99b]). Experimental evaluation by many authors and the theoretical analysis in [Ca99a] have shown that migration of the fittest individuals from one population to other populations to replace the least fittest individuals in these populations results in substantial speed-ups of the distributed GA compared to a sequential GA.

Several authors saw as reason for the good results produced by multi-deme GAs that there is more diversity in such systems. But it is obvious that migration of the fittest individuals has a high chance to reduce diversity. Nevertheless, diversity in a GA-based system is a feature that can on its own enhance the system performance. Outside of the context of distributed GAs, there have been attempts to modify GAs (resp. Genetic Programming) towards maintaining more diversity in a population. For example, in [dJ+01], diversity consideration is added as an additional objective, thus transforming a single-objective problem into a multi-objective one. But naturally, this means having to deal with multi-objective optimization and problems like loosing focus.

In this paper, we present an improvement to distributed GAs based on multi-demes and migration that tries to achieve an additional focus on diversity without weakening the focus on fitness within the individual demes. More precisely, we propose to include diversity considerations into the process of selecting the individuals for migration. Our guiding vision for the multi-deme GA is not a set of islands with very fit individuals swimming between them, but a set of herds with exchange of cattle being controlled by ranchers that take into account a lot of criteria for selecting what additional stock they want to buy for their herd. And for us, the two criteria these ranchers are using are fitness (naturally) and diversity of individuals.

Our experiments with several optimization problems from literature show that considering also diversity when selecting individuals for migration can substantially improve the time until the solution to a problem is found. We also observed that the variance in run time of several runs (for the same problem instance) is smaller, thus achieving a more stable behavior when using diversity in the selection.

This paper is organized as follows: after this introduction, we present in Section 2 an agent-based view on distributed GAs based on migration that reveals several features of such GAs that can be varied. Using the agent-based view instead of a biologically motivated one also allows us to point out additional instantiations for some features. In Section 3, we present the, in our opinion, most promising additional feature instantiation, namely using additional criteria for the selection of individuals for migration. In Section 4, we evaluate this additional instantiation for the usage of diversity. Finally, in Section 5, we conclude with remarks on future work.

2 Distributed GAs based on Migration

From the perspective of search in general, GAs (and most other evolutionary methods) can be classified as set-based searches, in contrast to tree-based search methods like Branch-and-Bound or graph-based methods like A*. In fact, GAs might be the most prominent representants of set-based search, with examples for set-based search outside of evolutionary methods being generating theorem proving methods like Resolution with Factorization (see [Ro65]).

The general idea of set-based search is to represent the search state as a subset of a set $\mathcal{F}$ that defines the possible elements of states. In a GA, the set $\mathcal{F}$ is the set of all possible individuals. Since a search process produces a sequence of states, the possible successor states of a state have to be defined and then a search control has to pick which possible successor really gets chosen in an actual search run. In set-based search, the possible successor states are the product of applying rules that take as premises a (sub)set of the current state and either remove elements or define (using the premises) new elements to be added or both. In GAs,

these rules obviously are the Genetic Operators with additional rules for removing one or several individuals from the current state, usually based on their fitness compared to the fitness of the remaining individuals.

To a given state in a set-based search, usually several of the rules to generate successor states are applicable. Even more, one rule often can be applied with different instantiations of the premises out of the current search state, so that the task of a search control is not only to decide which rule should be applied next, but also with what premises. For GAs, there are many different search control schemes that try to find the right mix of influence of the fitness of individuals and random effects in order to select the individuals in the premises of a rule.

While there are many ways how a set-based search can be distributed, the idea of having several set-based searches performed in parallel has been suggested by many authors. In fact, the so-called competition approach (see [Er92]) has also been applied to other search paradigms. And letting the different searches exchange information during the search run is an obvious *improvement of the competition approach* (see [De00]). For GAs, presenting distributed GAs based on improving on the competition approach traditionally made use of the multi-deme or island model that does not explicitly use a processor or agent performing a search process but takes a more population centered approach. In order to motivate our improvement of distribution approaches for GAs based on improving on the competition approach –that we will present in the next section– in the following, we will present an agent-based model for such distribution approaches (an instantiation of a similar presentation for distributed search in general from [De00]).

A distributed set-based search can be modeled as several search agents (usually each of them running a search process on its own processor) that cooperate with each other. Let $\mathcal{A}g$ = $\{\mathcal{A}g_1,...,\mathcal{A}g_p\}$ be the set of these search agents. Essentially, each of these agents performs a set-based search process, but they also should communicate with each other. We model communication in general by employing a so-called *communication structure* $\mathcal{K}om$ = (D_1,...,D_l). The D_i in $\mathcal{K}om$ define the possible content of data areas that can be manipulated and read by the search agents. More precisely, each agent $\mathcal{A}g_j$ has a so-called environment $\mathcal{E}nv_j$ that contains a subset of $\mathcal{K}om$ (describing the data areas of $\mathcal{K}om$ that the agent monitors) and a communication function mes_j that can change the value of data areas in $\mathcal{K}om$ whenever the agent's search process reaches a new state. Due to this, a search process determines its next state not only based on its current state but also based on the current values of the data areas in $\mathcal{K}om$ in its agent's environment. Note that a D_i can be an element of the environment of several agents. But the data areas can also be used to transfer information to only one or a few agents and they can be used to represent different kinds of information.

In addition to the search agents, we need two more agents $\mathcal{A}g_S$ and $\mathcal{A}g_E$, the start and end agent. The task of the start agent $\mathcal{A}g_S$ is to provide each of the search agents with a start state and a goal to fulfill. Naturally, for this it uses the given instance of the search problem to solve. In general, the end agent has to produce a result to the given problem instance out of the results of the search agents. For distributed search based on improving the competition approach, this means to select the best solution found by any of the search agents.

If we look at this general framework for describing (set-based) distributed search, then distributed systems based on improving the competition approach can differ in

- the realization of the start agent
- the realization of an individual search agent
- the types of information exchanged between the search agents
- the assignment of data areas of $\mathcal{K}om$ to search agents and the use a search agent makes of data in an area
- the communication functions of the search agents.

In multi-deme distributed GAs, usually the start agent produces for each agent a different start population out of the same set $\mathcal{F}$ of possible individuals (and all the search agents have only elements of this $\mathcal{F}$ in their states during search). The theoretical possibility of favoring a different subset of $\mathcal{F}$ for different search agents has, to our knowledge, so far not been tested (naturally, the selection of the subsets is the problem that would have to be solved).

The wide area of cooperative co-evolution of demes (see, for example, [Sm+92]) defined on different sets $\mathcal{F}$ is not an example for distributed search based on improving the competition approach, but has to be classified as an instance of distributed search based on dividing the given problem instance into instances of subproblems. These subproblems are then solved by different search agents. In order to guarantee that the subproblem instance solutions are compatible with each other, i.e. produce together a good solution to the initial problem instance, the fitness evaluation within an agent uses the best individual of each other agent, combines it with the own individual to evaluate and then computes a global fitness. It has been claimed that the different demes of such approaches achieve a niching effect that overall produces more diverse solutions to the problem. Since diversity of solutions is also our goal, we mention this concept, although it is not a distribution approach of the type we are interested in.

Obviously, it is necessary that the individual search agents do not do exactly the same state transitions and thus perform a lot of redundant work. There is a wide variety of methods how this can be achieved for multi-deme based distributed GAs. Firstly, with having already different start states for the different agents, the agents will produce somewhat different sequences of states, especially since a GA has also a random influence regarding the selection of successor states. But there are additional methods to have the agents differ:

- different parameter values (like population size, mutation probability, and so on) for the same basic GA (as first suggested by Tanese, see [Ta89])

- different fitness functions (naturally with some connection, see [Eb+97] for an example)
- different sets of Genetic Operators for different agents
- different basic schemes for selecting the next successor state, beyond using the same scheme with different parameter settings.

The last two methods move towards having somewhat heterogeneous agents (although real heterogeneous systems would combine GA-based agents with non-set-based search agents, see [DO99]).

Most commonly, multi-deme based distributed GAs exchange individuals between the agents with the intention to integrate the received individuals into the own population. But the potential spectrum of types of information that can be exchanged is much larger and defined with regard to two general decisions (see [DO99], again):

- positive information vs. negative information and
- information to be integrated into the search state vs. information to influence the search control

Migration is not the only example for positive information to be integrated into the search state. In [Pe96], a whole state can be given from one agent to another and in [De95], the best agent sends its whole (improved) state to all other agents. An example for positive control information is also given in [Pe96], where the control parameters of the best agents are adopted by the worst agents (with some mutations to avoid too much redundancy). The exchange of negative information has, so far, not been considered very much in GAs.

For the data areas in the environment of an agent, there are two extremes that can be found: an agent observes everything other agents communicate by having the whole $\mathcal{K}om$ in its environment (which resembles very much blackboard-based communication) or each agent has one dedicated area for each type of information in its environment (which models a message-passing architecture). And everything in between is also possible. Two data areas might allow exactly the same content, but the use an agent makes of what it reads there can be rather different. For example, an individual in a data area for positive information for the search state will simply be integrated into the search state, while the same individual in an area for negative control information might be used by the agent to determine similar individuals in its state that then will be less likely to be considered as premises for Genetic Operators.

In general, the usage an agent makes of new data in an area of $\mathcal{K}om$ naturally depends on the type of information. But also which agent has send the information can be considered and one potential "usage" is often not considered: throwing the information away without using it (see the concept of receive-referees in [DO99]). If information can be misleading or can slow down the search process, then thinking about throwing it away is a good idea. So, again, we have a spectrum ranging from letting data in data areas of $\mathcal{K}om$ override the agent's own data (for example using a search state put in such an area as next search state, without any connection to an agent's own search state, see [De95]), over adding/mixing received data and own data (as migration does) to ignoring received data.

The communication function of an agent is mostly responsible for selecting the information an agent should communicate to other agents (which includes deciding when selection and communication take place) and for determining to what other agents a particular information will be communicated. The later is the opposite side of a coin to determining which data areas an agent observes. If an agent $\mathcal{A}g_i$ does not observe all data areas in $\mathcal{K}om$, then the communication function mes_j of another agent $\mathcal{A}g_j$ can avoid sharing information with $\mathcal{A}g_i$ by writing the information into data areas not observed by $\mathcal{A}g_i$. Again, we have a spectrum of possible realizations of this part of the communication function ranging from sending information to one agent only, over a group of agents, to all other agents. If only a group of other agents is addressed, then in distributed GAs often the agents in this group are determined using a neighborhood relation (which was suggested as an interesting feature by Grefenstette, see [Gr95]).

Who receives a certain information naturally can have quite an influence on the selection process for the information. If the agents are rather different, then it is a good strategy to send to a particular agent information that is

a) difficult for this agent to obtain on its own and

b) very useful for the agent in its current state of its search.

To achieve this is rather difficult, since it requires not only general knowledge about the other agent, but also knowledge about its current needs. But we find at least some use of this realization possibility for mes-functions in [Pe96].

Nevertheless, most distributed GAs select information to be communicated without taking the receiving agents into account. Consequently, they base their selection decisions on criteria that can be measured using their search results so far, most often trying to measure the success a certain piece of information produced subsequently for the agent itself. In distributed GAs, an obvious success measure for individuals is their fitness. But fitness is by far not the only criterion that can be applied. There are criteria that are based on the problem to solve and criteria that are based on the search process used. An example for the later is using diversity of a solution (compared to the best solution of an agent), which we will define more precisely in the next section. Naturally, for different kinds of information different selection criteria can, resp. must, be employed.

As can be seen by our description so far, there are many features (with quite a lot of feature values) of a distributed search system that can be varied. Consequently, there is still a lot of evaluations to be done and this paper tackles only one of the possibilities.

3 Migration based on Fitness and Diversity

As stated at the end of the last section, there is a wide variety of actual distributed GAs that can be designed using the presented features of distributed GAs. If we add to this the also very large variety of sequential GAs that can be designed using features like representation of an individual, possible Genetic Operators, or selection strategies of the search control, then an evaluation of all features, their instantiations and the possible combinations is well beyond what can be covered even in a book. And due to the large variety of problems that can be solved using GAs and the often rather different behavior a search process shows for different instances of a single problem, experimental evaluations of distributed GAs (i.e. a concrete instantiation of all the features from the last section and of the features of sequential GAs that are used within the search agents) can easily provide experiences that are difficult to relate to the experiences reported by others.

For this paper, we have chosen to concentrate on one additional feature instantiation of the criteria the mes-functions of agents use to select individuals to be communicated to other agents: the use of diversity as additional criterion. To connect our experiments to other work, we have chosen as instantiations of the other features of distributed GAs the ones presented in [Ca99a] that presented the ground work on some other criteria of the selection of individuals (and on how the selected individuals are integrated in the search of the receiving agents).

More precisely, we investigate the following concrete distributed GA concept: The start agent $\mathcal{A}g_S$ creates for each search agent $\mathcal{A}g_i$ a start state $s_{0_i} = \{ind_{i1},...,ind_{ik}\}$ with $ind_{ij} \in \mathcal{F}$. Since we will have to measure the difference of elements in $\mathcal{F}$, we need to know how an element of $\mathcal{F}$ looks like. While we think that our concept works for all kinds of representations of individuals (naturally provided that a measure for diversity can be defined), our experimental evaluation considers only bitstrings (of length n) as representation for individuals. As usual, the end agent will select the best individual generated by any search agent during a run as the result of the run of the distributed GA.

Our search agents will all use the same search process (i.e. sequential GA), so that we rely on the different start states and the random influence in the search control to produce different sequences of search states in different agents. This means that the sequential GAs used by the agents use the same parameter settings, the same fitness function, the same set of Genetic Operators and search controls employing the same basic scheme for selecting the Genetic Operator to apply and the parent individuals required by the selected operator.

The only type of information the agents exchange is positive information to be included into the search state, namely individuals from the current search state (i.e. population) of a search agent. Each agent receives the same information, which can be either achieved by having one data area in $\mathcal{K}om$ that is in the environment of all agents (i.e. $\mathcal{K}om = (D_{common})$, with $D_{common} = \mathcal{F}^*$) or by having one data area for each agent and an agent writing the same information in the data area of each other agent (i.e. $\mathcal{K}om = (D_1,...,D_p)$, with $D_i = \mathcal{F}^*$). Since we want to use networks of machines to implement our distributed GA and therefore used in our implementation a modification of the TWlib (see [DL96]), we use the second alternative. A search agent integrates every new individual it finds in its data area into its current population, deleting the individuals in the population that have the worst fitness to make room for the new individuals (following the results of [Ca99a]).

The communication functions of the agents communicate the same information to all other agents, as already stated (i.e. mes_i adds the selected individuals to the current values of D_1 to D_{i-1} and D_{i+1} to D_p). For selecting the individuals to migrate, the mes-function of an agent computes for each individual ind in the current population that is not the best individual ind_{best}[1] a quality measure

$$qual(ind) = w_{fit} \times \frac{\text{fitness}(ind)}{\text{fitness}(ind_{best})} + w_{div} \times \frac{\text{difference}(ind, ind_{best})}{n}$$

and then selects the m-1 individuals with the highest $qual$-value and the best individual for migration.

In the formula above, w_{fit} and w_{div} are weights for the two criteria used for selection. We will evaluate different settings for them in our experiments. In order to have the influence of both criteria approximately equal (before using the weights) we normalize them by dividing by the fitness of the best individual, resp. by dividing by the length of the bitstring representing the individual.

This already hints at the particulars of our difference-function. Naturally, there are many possible functions that can be considered to indicate the difference between two individuals despite the fact that we already have limited ourselves to individuals represented by bitstrings. We tried to make our difference-function as simple as possible, especially we did not want to use anything related to the general problem to solve, and therefore we just count the bits in the two individuals in which they differ. Formally, for two individuals $a_1...a_n$ and $b_1...b_n$, the definition is

$$\text{difference}(a_1...a_n, b_1...b_n) = \sum_{i=1}^{n} (a_i \otimes b_i)$$

Here, $\otimes$ indicates the XOR-operator, so that our difference is the Hamming distance of the two individuals.

The final feature we have to specify is directly related to the selection process for migration, namely when this selection takes place. In theory, the mes-function could select individuals in every search state, but this would result in a lot of communication (note that even if it does not select anything, it either has to tell the other agents that this is the case or the other agents have continuously to check for any communication). Therefore we will have migration taking

[1] Here the best individual is the one with the highest fitness. If several individuals have the same fitness, one of them is randomly chosen as ind_{best}

place after a given number of generations (and this number is a parameter of the distributed GA).

In principle, the exchange of individuals can be performed synchronously (i.e. all agents interrupt their search, run their mes-functions, what until they have the results from everyone else and then integrate what they got from the others) or asynchronously (i.e. an agent interrupts its search, runs its mes-function, integrates whatever is new since the last time, and continues its search without waiting until it heard from every other agent). Since we use the TWlib as basis (which favors synchronous "team meetings"), we have chosen a synchronous exchange.

4 Experimental Evaluation

In this section, we report on our experiments we performed using the distributed GA concept (resp. improvement) from last section. The search problems we are using are taken from [DM02] (we modified some of them to either run longer or shorter to stay within the limits of running at least 1.5 minutes and at most 60 minutes), which presents optimization problems for Genetic Algorithms collected from several sources. The examples we picked are characterized in [DM02] as being difficult due to having many local optima in which GAs can become stuck, a problem we think can be avoided by using diversity in migration. And our experiments show that our belief is correct.

Before we present and analyze our experiments, we first have to present the basic sequential GA that is used by our search agents. We do not use exactly the algorithm of [Ca99a]. While we use the generation concept, for selecting the parent individuals to a crossover we use the wheel-of-fortune method to combine fitness with a random influence. This means that each individual's chance to be selected as parent is proportional to its fitness-value. With a certain chance the result of such a crossover is then mutated. A new generation is computed by generating the necessary number of new individuals, allowing all individuals of the previous generation as possible parents, and adding to these new individuals a certain number of the fittest individuals of the old generation. In our experiments, we always used 50 as the size of a population, with 5 individuals from the old generation surviving. The mutation rate was one percent.

The general setting of our system for all the experimental series was as follows: we had available 4 Sun blade machines and therefore we used 4 search agents. Migration takes place every 10 generations and each agent then selects m=5 individuals, which naturally means that the 15 worst individuals of every agent are then replaced.

We tried 4 different settings for the combination of the weight parameters w_{fit} and w_{div}, seeing them as percentages of influence: 100:0 (which means without using diversity at all), 70:30, 50:50 and 30:70. Obviously, the setting 100:0 represents the basic distributed GA without our improvement and will be our reference point. Each experimental series consists of 10 runs of the distributed GA for each of the 4 settings and will be reported in its own table. The table presents the best, worst and average run time until the optimal solution (that we got out of the paper mentioned above) occurs first in the population of one of the agents. For the readers convenience, we also report the difference in run time between best and worst run that allows us to judge the stability of the run time behavior of the different settings. The best result of a particular row is printed in bold.

Weight ratio:	100:0	70:30	50:50	30:70
time best run	311.1	170.4	181.5	**157.5**
time worst run	496.6	**218.0**	270.1	382.5
average time	384.2	**193.0**	215.4	247.2
worst - best	185.5	**47.6**	88.6	225.0

Table 1: Run time comparison different ratios of fitness vs. diversity for F_1 with q=25, times in seconds

Weight ratio:	100:0	70:30	50:50	30:70
time best run	566.9	**308.4**	321.0	314.3
time worst run	899.1	**447.0**	486.9	586.9
average time	748.8	**377.4**	399.2	430.8
worst - best	332.2	**138.6**	165.9	272.6

Table 2: Run time comparison different ratios of fitness vs. diversity for F_1 with q=40, times in seconds

We have chosen 4 different general function schemes for our tests. The schemes allow for different numbers of input variables (that have a limited range that is encoded by an appropriate number of bits in an individual) thus producing a variety of lengths for individuals. More precisely, the variables for the functions (x_1,...,x_q in the following definitions) are floating point numbers. The following schema was used to represent the variables in an individual. Each variable is represented by 10 consecutive bits in the individual. So if there are 40 variables to the function the individuals in the population are 400 bits long. Representation of the floating point numbers for each variable was done in the following way. Since each variable is 10 bits long we have the possibility to represent 1024 numbers. To convert the bits in the variable to a floating point number a conversion table of 1024 elements was created. The range of floating point numbers was between 10.0 and -10.0. This means that the increment between each spot in the conversion table is of the size ((20.0)/1024) = 0.01953. For example the parameter 0000000000 gets mapped to 0 and the parameter 0000000001 gets mapped to 0.01953. With this in mind, crossover is done through single point crossover and mutation involves the negation of bits in the individual.

The first 6 experimental series will explore 3 different numbers of variables for two schemes, while the other 2 series explore one variable number for the 2 other schemes (we substituted the chosen number already into the formulas).

The function scheme for the first 3 series is

$$F_1(x_1, ..., x_q) \quad = \quad \sum_{i=1}^{q} x_i^2$$

Table 1 reports the results for q=25, Table 2 for q=40 and Table 3 for q=50. The best average run time was al-

Weight ratio:	100:0	70:30	50:50	30:70
time best run	1040.8	**430.4**	582.2	515.5
time worst run	—	**734.5**	798.8	936.6
average time	1274.4	**554.6**	691.2	740.2
worst - best	—	304.1	**216.6**	421.1

Table 3: Run time comparison different ratios of fitness vs. diversity for F_1 with q=50, times in seconds

Weight ratio:	100:0	70:30	50:50	30:70
time best run	55.9	44.6	33.6	**31.3**
time worst run	100.7	66.4	**55.0**	55.5
average time	74.7	54.2	44.7	**43.6**
worst - best	44.8	21.8	**21.4**	24.2

Table 4: Run time comparison different ratios of fitness vs. diversity for F_2 with q=30, times in seconds

Weight ratio:	100:0	70:30	50:50	30:70
time best run	100.9	79.2	**55.2**	58.0
time worst run	205.3	137.4	100.4	**99.9**
average time	155.0	103.7	**77.5**	77.9
worst - best	104.4	58.2	45.2	**41.9**

Table 5: Run time comparison different ratios of fitness vs. diversity for F_2 with q=40, times in seconds

Weight ratio:	100:0	70:30	50:50	30:70
time best run	154.8	108.2	105.0	**88.5**
time worst run	322.8	177.4	**157.5**	165.7
average time	234.7	137.1	**127.5**	130.4
worst - best	168.0	69.2	**52.5**	77.2

Table 6: Run time comparison different ratios of fitness vs. diversity for F_2 with q=50, times in seconds

ways produced by having 30 percent influence of the diversity measure and the improvement is really big, cutting the run time in half. But also the other settings are better than the 100:0 setting, showing that adding diversity to migration pays off. The worst run with the 70:30 setting is for all 3 series better than the best one without using diversity, which is also true for the 50:50 setting. In fact, for the 70:30 and 50:50 settings the range of results (see last row of the tables) is substantially smaller than what is achieved by using the fitness only. For q=50 (Table 3) only half of the runs for 100:0 finished within the allotted time and the average was computed using the 5 successful runs only. It should be noted that in Table 1 only the worst run for the 30:70 ratio was worse than the best run without using diversity. In Table 2, the same is true (again, only the worst run for the 30:70 setting). And in Table 3, all runs using some influence of diversity were better than the best run of the 100:0 setting.

The second function scheme is

$$F_2(x_1, ..., x_q) \quad = \quad 200 + \sum_{i=1}^{q} (x_i^2 - 10\cos(2\pi x_i))$$

The three values for q reported in Tables 4, 5, and 6 are 30, 40, and 50. As these tables show, a larger influence of diversity than for the first function scheme produces the best results. While sometimes 50:50 and sometimes 30:70 is a little bit better, nevertheless the differences are usually very small (especially with regard to average run time). Both influence settings achieve huge improvements against the runs without using diversity, again, and especially the range of run times is greatly reduced. This is also true for the 70:30 setting, where also the difference between worst and best run is nearly half of what the runs without using diversity produced. It should be noted that, while not as good as for 50:50 and 30:70, the results for a ratio of 70:30 in all categories are still a lot better than the runs without diversity, so that for this function scheme the usage of diversity is very successful, again. This is also indicated by all runs for the 50:50 and 30:70 ratios for q=30 and q=40 being better than the best run without using diversity. For q=50, only one of the runs for each of these two ratios was slower than the best run without using diversity.

The third function scheme is

$$F_3(x_1, ..., x_{10}) \quad = \quad 1 + \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos(\frac{x_i}{\sqrt{i}})$$

Table 7 shows that while still the average run time of all settings using diversity is better than the reference case, the improvement is less than for the previous experimental series. Also the range of run times is not as much smaller as in the first 6 series. But still, we clearly have an improvement.

The final tested function scheme is

$$F_4(x_1, ..., x_{40}) \quad = \quad 41898.29101 + \sum_{i=1}^{40} (-x_i \times \sin(sqrt|x_i|))$$

As Table 8 shows, only the average run time of the setting 30:70 is better than 100:0, although 50:50 is best in the best and worst run time category. But still, the range of results is substantially smaller for all settings with diversity than for using just fitness for selection and the average run time is for 70:30 and 50:50 not much worse than for 100:0.

All four function schemes were selected, since they produce a large number of local optima making it likely for a GA to be trapped for some time in one that is not the global optimal one. The migration of individuals that are also selected based on diversity reduces the likelyhood for this to happen significantly, while the fact that each agent still concentrates mostly on the fittest individuals pushes the search for the optima forward.

This means that the usage of diversity in migration should not be very successful for optimization functions that have only a few local optima. In fact, we might even slow down the search due to exchanging mostly individuals that are less near to the global optimum than the fittest individuals are (if we look at the analysis in [Ca99a]). Therefore we conducted an experiment using the so-called trap functions that were also used in [Ca99a]. Trap functions have only

Weight ratio:	100:0	70:30	50:50	30:70
time best run	312.7	265.9	**239.9**	271.8
time worst run	650.1	510.2	**485.7**	567.5
average time	470.9	**361.7**	375.6	391.7
worst - best	337.4	**244.3**	245.8	295.7

Table 7: Run time comparison different ratios of fitness vs. diversity for F_3, times in seconds

Weight ratio:	100:0	70:30	50:50	30:70
time best run	332.5	380.0	**317.5**	346.2
time worst run	780.3	685.5	**650.0**	674.5
average time	526.7	529.9	532.6	**491.6**
worst - best	447.8	**305.5**	332.5	328.3

Table 8: Run time comparison different ratios of fitness vs. diversity for F_4, times in seconds

one global and one additional local optimum and the populations produced by a GA of the type we are using in our agents often are "trapped" for quite some time in the local optimum before finally finding the global one. As Table 9 shows, as we expected, enhancing the selection of individuals for migration by diversity does not pay off for such functions.

So, including diversity as an additional criterion for selecting individuals that migrate in multi-deme distributed GAs improves the average run time for problems that have a lot of local optima. For these problems, it also makes the performance of the system much more stable and hence provides better predictability. While weighting fitness and diversity equally seems to be in most cases a good strategy, varying the weighting of fitness versus diversity can improve the results quite a bit.

5 Conclusion and Future Work

We presented an improvement to multi-deme based distributed GAs that selects individuals for migration not only based on fitness but also takes diversity between individuals into account. Our first experiments with problems with lots of local optima showed that simply using the Hamming distance as diversity not only leads to finding the global optimum faster, but also produces a smaller range of the times of repeated runs to the same problem instance, thus making the system more stable and predictable, thus improving the positive influence of multi-deme based GAs reported by other authors.

In addition to a much more extensive experimental evaluation, in the future we want to develop other possible criteria for the selection of individuals to migrate, that could improve the performance of distributed GAs. This can start with other, more complex diversity measures (for example a cascading selection that measures diversity of an individual to all individuals an agent has already selected for migration) and diversity measures for individuals that are not represented as bitstrings. The agent-based view on distributed GAs we presented also identified additional distributed GA

Weight ratio:	100:0	70:30	50:50	30:70
time best run	**709.6**	714.2	735.0	736.8
time worst run	**874.3**	995.4	878.1	971,7
average time	**761.5**	870.8	826.3	871.6
worst - best	164.7	281.2	**143.1**	234.9

Table 9: Run time comparison different ratios of fitness vs. diversity for a trap function, times in seconds

variants that have to be examined. Some of those also should profit from including diversity. Examples are using diversity to determine the quality of a whole population or for selecting negative information.

Acknowledgements

This work was supported by NSERC under grant RGP 238831.

Bibliography

[Ca99a] Cantu-Paz, E. (1999) "Migration Policies, selection Pressure, and Parallel Evolutionary Algorithms", IlliGAL Report 99015, University of Illinois.

[Ca99b] Cantu-Paz, E. (1999) "Designing Efficient and Accurate Parallel Genetic Algorithms", IlliGAL Report 99017, University of Illinois.

[De95] Denzinger, J. (1995) "Knowledge-Based Distributed Search Using Teamwork", Proc. ICMAS-95, San Francisco, pp. 81–88.

[De00] Denzinger, J. (2000) "Conflict Handling in Collaborative Search", in Tessier, Chaudron, Mller (eds.): Conflicting Agents: Conflict management in multi-agent systems, Kluwer, pp. 251–278.

[dJ+01] de Jong, E.D.; Watson, R.A. and Pollack, J.B. (2001) "Reducing Bloat and Promoting Diversity using Multi-Objective Methods", Proc. GECCO-01, San Francisco, pp. 11–18.

[DL96] Denzinger, J. and Lind, J. (1996) "TWlib - a Library for Distributed Search Applications", Proc. ICS'96-AI, Kaohsiung, pp. 101–108.

[DM02] Digalakis, J.G. and Margaritis, K.G. (2002) "An Experimental Study of Benchmarking Functions for Genetic Algorithms", Intern. J. Computer Math. 79(4), pp. 403–416.

[DO99] Denzinger, J. and Offermann, T. (1999) "On Cooperation between Evolutionary Algorithms and other Search Paradigms", Proc. CEC-99, Washington, pp. 2317–2324.

[Eb+97] Eby, D.; Averill, R.C.; Gelfand, B.; Punch, W.F.; Mathews, O. and Goodman, E.D. (1997) "An Injection Island GA for Flywheel Design Optimization", Proc. EUFIT'97.

[Er92] Ertel, W. (1992) ”Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen”, PhD thesis, Technical University of Munich.

[Gr85] Grosso, P.B. (1985) ”Computer simulations of genetic adaptation: Parallel sub-component interaction in a multilocus model”, PhD thesis, The University of Michigan.

[Gr95] Grefenstette, J.J. (1995) ”Robot learning with parallel genetic algorithms on networked computers”, Proc. SCSC 95, Ottawa, pp. 352–357.

[Pe96] Pedroso, J.P. (1996) ”Niche Search: An Evolutionary Algorithm for Global Optimization”, Proc. PPSN IV, Berlin, LNCS 1141, pp. 430–440.

[Ro65] Robinson, J.A. (1965) ”A Machine Oriented Logic based on the Resolution Principle”, JACM 12, 1, pp. 23–41.

[Sm+92] Smith, R.E.; Forrest, S. and Perelson, A.S. (1992) ”Searching for diverse, cooperative populations with genetic algorithms”, Evolutionary Computation 1, 2, pp. 127–149.

[Ta89] Tanese, R. (1989) ”Distributed genetic algorithms for function optimization”, PhD thesis, University of Michigan, Ann Arbor.