Computer Science • 14 (1) 2013 http://dx.doi.org/10.7494/csci.2013.14.1.77

Daniel Krzywicki

NICHING IN EVOLUTIONARY MULTI-AGENT SYSTEMS

Abstract *Niching is a group of techniques used in evolutionary algorithms, useful in several types of problems, including multimodal or nonstationary optimization. This paper investigates the applicability of these methods to evolutionary multi-agent systems (EMAS), a hybrid model combining the advantages of evolutionary algorithms and multi-agent systems. This could increase the efficiency of this type of algorithms and allow to apply them to a wider class of problems. As a starting point, a simple but flexible EMAS framework is proposed. Then, it is shown how to extend this framework in order to introduce niching, by adapting two classical niching methods. Finally, preliminary experimental results show the efficiency and the simultaneous discovery of multiple optima by this modified EMAS.*

Keywords niching, evolutionary algorithms, multi-agent systems

1. Introduction

Niching is a group of techniques used in evolutionary computation. They are helpful in solving multimodal optimization problems and also allow to achieve better results, both in single and multi-criteria optimization.

The idea of introducing niching mechanisms into classical evolutionary algorithms has already been broadly studied. A number of such techniques has been proposed and their effectiveness examined [4, 14, 15, 19].

However, the applicability of niching techniques has not been quite so much investigated when it comes to evolutionary multi-agent systems (EMAS) [3]. EMAS combines the advantages of evolutionary algorithms and multi-agent systems. By decentralizing the logic of the algorithm into the autonomous individuals of a MAS, selective pressure emerges instead of being designed at a population level, the population size can freely fluctuate and the whole algorithm is much more extensible and scalable.

Developing niching methods for EMAS is important, as it opens the possibility of using them in a much wider class of problems. It might also increase their efficiency in practical business problems which tend to be computationally difficult and highly multimodal.

An analysis of such applicability, though restricted to coevolutionary multi-agent algorithms, has been performed in [6]. However, it is still an open question to what extent these techniques can be used in simpler and more general EMAS.

The use of niching techniques in MAS has also been considered in [5]. However, the authors focused on using them to reduce the product space (i.e. the set of possible combinations of agents actions) in a cooperative Markov Decision Problem. Instead, this paper considers niching as it understood in the field of evolutionary algorithms, i.e. as a mean of dividing the solution space (i.e. the set of possible solutions to an optimization problem).

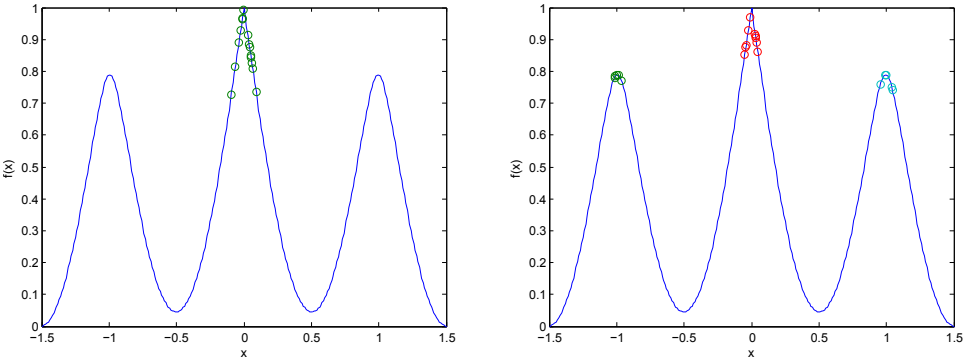
The purpose of this paper is thus to show that existing niching techniques, designed for classical evolutionary algorithms, can also be adapted to be used in an evolutionary multi-agent system.

As a starting point, this papers begins by reviewing in section 2 the concept of niching and describing two chosen classical methods: crowding and the clearing procedure. Section 3 formulates a simple but flexible EMAS framework, as a basis for further extension. Then, it is shown in section 4 how to adapt these two classical niching techniques into such an EMAS framework. Finally, the capabilities of these techniques are examined in section 5, along with preliminary experimental results.

2. Niching in classical evolutionary algorithms

A basic evolutionary algorithm in general finds only one solution to an optimization problem. It is thus not suitable for multimodal optimization, in which the goal is to find many such optima. In order to change that, niching is introduced.

The idea of niching is based on the natural observation that there exists no globally optimal super-species (certainly not *homo sapiens*). Instead, a large number of species co-exist, each one locally optimal in its immediate ecological surrounding, which is called a *niche*. Reproduction is usually restricted to within a species, competition to within a niche. Both concepts (species and niches) are thus two sides of the same coin and express some *locality of interactions* happening in the natural world [11].



(a) without niching (b) with niching

Figure 1. Results of multimodal optimization with and without niching. Values of x are potential solutions to some optimization problem and *f(x)* is the associated fitness.

Such a locality of interactions, though differently interpreted, is the basis for most niching techniques in evolutionary algorithms.

In the context of optimization, niches are identified with separate local optima. More accurately, with their *attraction basins*, i.e. the set of all initial points which have a given optimum as the final fixed point of a naive hill-climbing method.

Niching thus allows the simultaneous search of several areas in the solution space, which is essential in multimodal optimization. It also prevents the premature loss of genetic diversity, as the population is spread across the surroundings of several optima. This in turn is helpful in unimodal optimization, as it avoids getting stuck in only one local optimum in the long run. Niching is also useful in the case of nonstationary problems, when currently local optima can turn out to be global later on. It is thus efficient to track them down early on, precisely what niching does.

In the paper [14] two basic classes of niching methods have been defined: a) temporal (sequential) niching, and b) spatial (parallel) one. Sequential niching consist of repeatedly starting computations with the hope of finding a different result each time. Parallel niching, in turn, creates and maintains niches in the population itself, during one computation. Since most sequential niching runs might often end with the same results, parallel niching is mostly used in practice.

Well-designed niching techniques should meet the following criteria:

1. Niche capacity should be proportional to the quality of the corresponding optimum. In other words, better optima should be colonized by a greater number of individuals.
2. The algorithm should guarantee a stable existence of already created niches. Niches lifetime should thus at least grow exponentially with their size.

A comprehensive review of existing niching techniques has been presented in [6]. They have been divided into three classes of algorithms, based on:

1. Modified succession rule (i.e. preselection, crowding factor model, deterministic and probabilistic crowding, restricted tournament selection).
2. Modified selection rule (i.e. fitness sharing, clearing procedure, sexual selection).
3. Limited range of selection and reproduction, achieved by grouping individuals or introducing some topology in the environment (i.e. multinational evolutionary algorithm, population clustering, parallel evolutionary algorithms).

For the purposes of this paper, a few simple but popular and efficient methods have been chosen and described hereinafter: *crowding* and the *clearing procedure*.

2.1. Crowding

Deterministic crowding has been presented in [14] and extends the crowding factor model [4]. It is based on the assumption that individuals are most similar to their parents and should compete with them for their place in the population.

Crowding combines the mechanisms of selection and reproduction, similarly to tournament selection. Individuals from the population are randomly paired and reproduced. Offspring genotypes are recombined with probability P_C and mutated with probability P_M.

The crowding technique simulates the limited resources of the environment by assuming a fixed population size. Each child must thus compete with one parent for his place in the population. Assuming that $d(i_1, i_2)$ is a measure of the difference between individuals i_1 and i_2, parents and offspring are paired so as to minimize the expression $d(c_i, p_i) + d(c_j, p_j)$. In other words, it maximizes the similarity between competing parents and offspring. The winner of such a competition is chosen according to some *replacement rule*.

According to the *deterministic replacement rule* the winner is always the fitter individual. The probability $P(c)$ that the child c replaces the parent p is given by the eq. 1. The function f is the fitness of individuals.

$$P(c) = \begin{cases} 1 & \text{if } f(c) > f(p) \\ 0.5 & \text{if } f(c) = f(p) \\ 0 & \text{if } f(c) < f(p) \end{cases} \tag{1}$$

Deterministic rule of succession is very simple, but has a drawback: lack of so called *restorative pressure* [16, 17] – less fit niches tend to disappear with time in

favor of those better fit – event if the difference in fitness is very small. To prevent this, a *probabilistic replacement rule* have been proposed [16], where the probability of winning depends on the relative fitness of the individuals (eq. 2).

$$P(c) = \frac{f(c)}{f(c) + f(p)} \tag{2}$$

This concept have been extended in [15] into a *generalized replacement rule*. It consists of adding an additional parameter $\phi \in \Re^+ \cup \{0\}$ and the following probability (eq. 3):

$$P(c) = \begin{cases} \dfrac{f(c)}{f(c) + \phi \times f(p)} & \text{if } f(c) > f(p) \\ 0.5 & \text{if } f(c) = f(p) \\ \dfrac{\phi \times f(c)}{\phi \times f(c) + f(p)} & \text{if } f(c) < f(p) \end{cases} \tag{3}$$

In particular, for $\phi = 0$ generalized crowding is equivalent to deterministic one, and for $\phi = 1$ to its probabilistic version. Values in the range $0 < \phi < 1$ allow a more gradual adjustment of the probability of having the less fit individual winning, independently of the actual fitness function. Interestingly, values of $\phi > 1$ allowed to achieve good results for some fitness functions [15].

Regardless of the replacement rule, the crowding technique is very fast and easy to use. The big advantage is that there is no parameter directly associated with the fitness function or the amount of niches sought. Restorative pressure can be amplified by introducing a probabilistic replacement rule, whose effect can be graduated in the generalized version.

2.2. Clearing procedure

The clearing procedure [19] is based on a modification of the fitness function before selection and reproduction happen. It assumes the decomposition of the population into S distinct subpopulations – niches. Within each subpopulation a best fit *dominating individual* is designated. All other individuals in the subpopulation have their fitness set to some minimum (in the case of a maximization problem), which effectively excludes them from further competition and reproduction. In other words, this algorithm assigns all the resources within a niche to one individual.

It is possible to generalize the algorithm by considering more than one dominating individual. The maximum number of such individuals is then called the *capacity* of a niche. In the extreme case when the capacity is equal to the population size, the clearing procedure does not change the evolutionary algorithm.

Subpopulations are determined according to some measure of difference between individuals δ. An individual x belongs to a niche, if the distance to the dominating individual is lower than some given *niche radius* R (Fig. 2). Such a radius is a parameter of this algorithm.

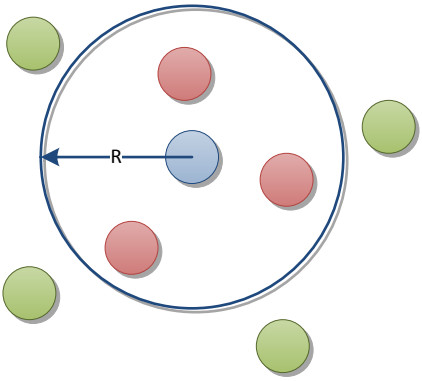


Figure 2. An example of the clearing procedure: the circles represent individuals in the solution space. The blue circle is a best fit individual which dominates red ones within a niche radius R. Green circles represent individuals unaffected by the blue one.

The partition of the population is performed as follows: individuals are sorted in a descending order of fitness. Starting from the best, each individual x *clears* a surrounding of radius R. Each individual y for which $\delta(x, y) < R$ is excluded from further reproduction. What is important, excluded individuals themselves do not take further part in the process – this clearing relation is not transitive. Such an exclusion simply consists in modifying the fitness of dominated individuals, usually by setting them to some minimum (in the case of a maximization problem).

The clearing procedure is very efficient. It is endowed with a strong restorative pressure as created niches persist for a long time. It is also relatively fast, its expected computational complexity is $O(|S|n)$, where$|S|$ is the number of dynamically designated niches (its pessimistic complexity is $O(n^2)$, though, but such cases can only happen in the early stages of the computation).

The drawback of this method is, however, the need to estimate an appropriate radius niche. This task is particularly difficult when the form of the fitness function is not known. However, there exist heuristic methods for estimating an efficient niche radius [20]. It is also possible to partition the population using hierarchical clustering techniques [20]. This allows to achieve a computational complexity of $O(n \log(n))$ and also removes the need for a niche radius parameter.

3. A framework for a Evolutionary Multi-Agent System

Evolutionary Multi-Agent Systems [3] combine these two concepts: evolutionary algorithms and multi-agent systems. Instead of considering a whole population in a centralized manner, the focus is on the individuals of which it is composed.

These identified individuals are assigned some information representing a solution to the problem and the associated fitness. They are also given some resources and au-

tonomy in action. They interact with each other in a bilateral way, share information through reproduction and compete over limited resources. Through these decentralized interactions, selective pressure *emerges* and drives the evolution forward. This bottom-up approach is easier to design and very scalable.

One could ask why to create yet another variation of an evolutionary algorithm. It turns out that many research and engineering areas are being currently revisited from a decentralized multi-agent perspective, with promising results [12]. The question of how complex evolutionary patterns emerge from simple and decentralized agent rules and interactions is itself worth being investigated.

Furthermore, for computational and optimization purposes, evolutionary multi-agent systems offer a few noticeable advantages over classical, centralized evolutionary approaches.

The first benefit is that agents, as they are individually identified and autonomous, do not have to die after reproducing. Parent agents usually share some of their resources with children ones, then continue acting like before and may live on for a long time. This is different from various crowding factor or elitist models, where only a given fraction of the population survives into the next generation. Instead, several generations may coexist and information can be shared between them.

Therefore, the population size no longer needs to be fixed and can instead freely fluctuate with the agents' births and deaths. The maximal number of agents only depends on the total amount of resources available in their environment. By changing that amount, one can also affect the number of agents and freely modify the size of the population, even during an ongoing computation. One possible application is to adapt the size of the population to the difficulty of the problem or to the size of the problem space.

Another consequence of these gradual generational changes is that the population varies much less abruptly. Because of that, many computationally intensive algorithms, like clustering, can be implemented in an efficient, incremental way. Therefore, even though one step of a classical evolutionary algorithm usually corresponds to multiple steps of an evolutionary multi-agent system, these can be much faster.

Evolutionary multi-agent systems have been successfully applied to such problems as portfolio optimization [8], investment strategies generation [7] or machine learning [1].

3.1. Architecture

The concept of evolutionary multi-agent systems is simple but very general, and there exist many different implementations. For the purpose of this paper, a possibly abstract, simple yet extensible architecture is hereinafter proposed.

3.1.1. Solution

Agents are assigned some information, representing a solution to the problem. This information can be in the form a binary genotype, a vector of real-valued numbers,

or have any other representation. This representation only needs to be coupled with the operators being later used, like crossover and/or mutation.

3.1.2. Fitness

Regardless of its representation, the evaluation of an agent's solution results in some fitness. However, agents are assigned two fitness values: an *objective* and an *effective* one. The objective fitness is based on the solution's evaluation and does not change during the agent lifetime. The effective fitness is based on the objective one, but also reflects the influence of the environment and elapsing time on the agent, in other words – the context. The effective fitness may very well change during the computation.

The objective fitness is important mainly for statistics, visualization and the end result. However, the algorithm should take into account the effective fitness of agents, and if necessary, modify it.

3.1.3. Energy

The decentralized nature of multi-agent systems requires a different selection mechanism than those used in classical evolutionary algorithms. Selective pressure should not be designed at the level of a whole population, but emerge in the result of bilateral interaction between agents.

This is usually achieved by introducing an explicit competition over limited resources. In this case, agents are given some *energy*. The total amount of energy in the system is fixed (though it might be changed by the user).

The reproductive ability of an agent needs only to depend on the amount of energy it possesses. Selection mechanisms may therefore favor better agents at the expense of worse ones, transferring energy from the former to the latter ones. Reproduction, as well as other custom actions, may require some amount of energy. Parent agents can share they energy with their offspring. Other agents can fight for energy. If its energy drops to zero, an agent dies.

3.1.4. Actions

Some versions [6] of evolutionary multi-agents systems allow complete autonomy of the agents – these can have multiple behavioral profiles, which are activated by some situations. For example, an agent with low energy will hunt for weaker individuals, while an agent with high energy will look for a mate for reproduction.

For the purposes of generality, however, this paper adopts a simplified model. Each agent has only one, common behavioral profile, which consists in a sequence of *actions*. Examples of such actions are fights, reproduction or death. These actions are performed sequentially and alternately, i.e. all agents execute the first action, then all agents execute the second action, etc.

The execution of all actions on all agents represent a *step* of the algorithm. In order the preserve consistency in the execution of actions, the addition or removal of agents should be differed until the end of a step.

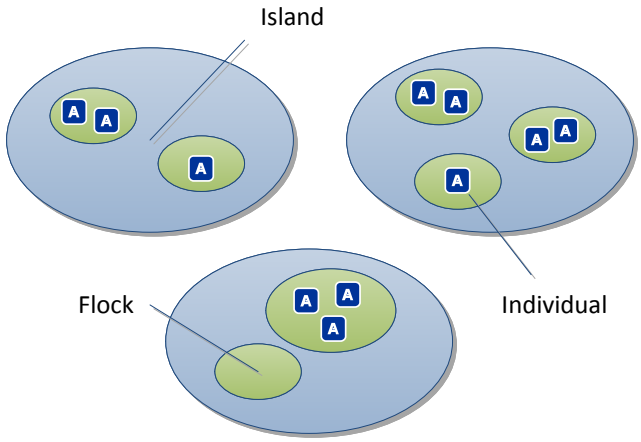


Figure 3. An example of environments structure: individual are grouped into flocks, which themselves are grouped into islands. This creates a tree-like hierarchy in which entities at each level can be treated as actual agents.

3.1.5. Environments

Agents are grouped into an environment. However, there can be more than just one environment, and these can have *structure* and *topology*. Structure means that individual agents can be for example grouped into flocks, which can be grouped into islands, and so on, resulting in a tree-like hierarchy (Fig. 3).

What is more interesting is that each entity in this hierarchy can itself be treated as an agent, with some properties or resources. This also allows to perform actions on environments, that is on whole populations.

Topology in turn means that equivalent entities (individual agents, islands) always have to be visible to each other. E.g. islands might be interconnected in a non-trivial pattern, while individual agents might live on a grid which restricts their movements and interactions. By default, however, all environments are assumed flat and fully connected.

3.2. Algorithmic skeleton

A skeleton of the algorithm executed by the evolutionary multi-agent system is proposed below. It consists in a sequence of *actions* which are executed by agents and represents their behavioral profile. These actions in turn delegate some implementation details to *strategies*.

This double decomposition offers great flexibility and extensibility to the algorithm. It is sufficiently general to be open to any customization and chosen actions or strategies can be modified or substituted without affecting other parts of the algorithm. Also, designing agents behaviors as a sequence of actions facilitates the

formal analysis of the correctness of an EMAS [2] as well as lead to efficient parallel implementations [21].

As a basis for further analysis, this paper adopts the following sequence of actions:

1. selection through battles;
2. sexual reproduction;
3. asexual reproduction;
4. death.

3.2.1. Battles

Agents are put into a competition for the available energy. Agents meet and then fight. Each such battle ends with the winner taking some energy from the loser. Such an action is presented in algorithm 1.

As it can be seen, the question of how agents meet, how much of such meetings happen, how a winner is chosen and how much energy is taken are left to strategies. As a reasonable default, this paper assumes single random encounters, deterministic fitness fights and a constant energy transfer.

Algorithm 1 BATTLE ACTION

```
1: procedure ExecuteOn(agent)
2:     for other ∈ EncounterWith(agent) do
3:         winner, loser ← Fight(agent, other)
4:         TransferEnergy(loser, winner)
5:     end for
6: end procedure
```

3.2.2. Reproduction

Agents can reproduce sexually and asexually. Sexual reproduction occurs first and obviously involves a pair of agents. As shown in algorithm 2, if both agents satisfy some predicate, reproduction happens. The resulting child agent receives some energy from parent ones. Each such child will be added to the environment at the end of the current step (see 3.1.4).

A default meeting strategy use again single random encounters. The reproduction predicate requires a sufficient amount of energy. The actual reproduction strategy depends on the representation of the agent's solution (see 3.1.1). Parent agents transfer a given proportion of their energy to the child agent.

Asexual reproduction happens next and is very similar, but does not involve meetings. The introduction of asexual reproduction allows to keep the population size reasonably big, as agents can't gather too much energy for a too long time. In the extreme case of one agent dominating and killing out all others, asexual reproduction can generate a new population and continue the calculation, though with a decrease in diversity.

Algorithm 2 Sexual Reproduction Action

```
1: procedure ExecuteOn(agent)
2:     for other ∈ EncounterWith(agent) do
3:         if Predicate(agent) && Predicate(other) then
4:             child ← ReproduceSexually(agent, other)
5:             TransferEnergy(agent, child)
6:             TransferEnergy(other, child)
7:             BufferedAdd(child)
8:         end if
9:     end for
10: end procedure
```

Just like in classical evolutionary algorithms, sexual reproduction usually involves using crossover and/or mutation operators, while asexual reproduction only use mutations.

3.2.3. Death

Agents simply die out if their energy drops to zero at the end of the step and are then removed from the environment.

4. Adapting niching techniques

In order to introduce niching into evolutionary multi-agent systems, one must consider how to fit them in the existing structure of the algorithm. The system could always be redesigned in such a way as to dedicate it to one of the techniques described in section 2. However, solutions which could introduce niching in a transparent way would have a much greater value.

The system described in section 3 can be extended as follows:

Agents
: Agents can be assigned additional properties or resources. It is a very transparent solution, as these additional features can be simply ignored by the rest of the system.

Actions
: It is also possible to define new actions, both at the level of individual agents and whole environments. Another solution is the modification of existing actions. While the first case may be transparent, the latter implies a much greater interference with the algorithm.

Strategies
: The last, most natural solution is a suitable implementation of some of the basic strategies mentioned in section 3. This is indeed the essence of that design pattern.

Below is shown how classical niching techniques, described in section 2, can be adapted to be used in an EMAS.

4.1. Crowding

The main problem with using crowding in EMAS is its assumption that the offspring competes with parents for limited space in the population. It implies that the size of the population has to be constant, which is in contradiction with one of the basic features of EMAS.

In classical evolutionary algorithms, crowding combines the selection and reproduction mechanisms. Similarly, to apply this technique directly to EMAS, a dedicated action is needed, which will replace both the battle (selection) and the reproduction, as shown in algorithm 3.

Algorithm 3 CROWDING ACTION

```
 1: procedure ExecuteOn(agent)
 2:     for other ∈ EncounterWith(agent) do
 3:         if Predicate(agent) && Predicate(other) then
 4:             child ← ReproduceSexually(agent, other)

 5:             if Distance(child, agent) < Distance(child, other) then
 6:                 closestParent ← agent
 7:             else
 8:                 closestParent ← other
 9:             end if

10:             winner ← Fight(child, closestParent)
11:             if winner = child then
12:                 TransferAllEnergy(closestParent, child)
13:                 BufferedAdd(child)
14:             end if
15:         end if
16:     end for
17: end procedure
```

As in the previous reproduction action, a strategy is used(line 4). Then the parent most similar to the child agent is selected and competes with the child 10). The fight can be deterministic, probabilistic or generalized, just like replacement rules in classical crowding. If the child agent wins, it takes all energy from its parent and gets added to the environment (line 13). If the winner is the parent, the new agent is never added to the environment and discarded instead.

If a parent agent loses the competition, it should not be allowed to reproduce again. Otherwise, a few new agents could replace the same parent, violating the assumption of a constant population size.

Energy loses its meaning, since all agents reproduce at least once, regardless of an energy predicate. The sole remaining purpose of energy is to indicate that an agent should die and prevent it from reproducing again. It is sufficient to assign the initial population with any positive energy. The predicate conditioning reproduction should therefore only check whether the energy of agents is still positive. Losing agents have their energy taken away, which excludes them from further reproduction and have them removed from the environment in the death action.

The advantage of this technique is its decentralized nature, based on bilateral interactions between agents, which is consistent with the spirit of EMAS. There is also no parameter, like in classical crowding. However, the are two much more serious disadvantages: a constant population size and a useless energy resource, so that the main benefits of using EMAS are lost.

4.2. Clearing procedure

The clearing procedure can be very easily applied to an EMAS, by leveraging the existence of two distinct fitness values – an objective and an effective one. The latter can be modified to reflect the dominance of strong agents other weaker and similar ones.

Algorithm 4 Clearing Action

```
1: procedure ExecuteOn(island)
2:     agents ← Individuals(island)
3:     agents ← NewQueue(agents)
4:     DescendingObjectiveFitnessSort(agents)

5:     worseAgent ← Last(agents)
6:     minimalFitness ← ObjectiveFitness(worseAgent)

7:     while agents is not empty do
8:         first ← Take(agents)
9:         EffectiveFitness(first) ← ObjectiveFitness(first)

10:        for other ∈ agents do
11:            if Distance(first, other) < radius then
12:                EffectiveFitness(other) ← minimalFitness
13:                Remove(agents, other)
14:            end if
15:        end for
16:    end while
17: end procedure
```

The algorithm 4 shows the clearing action to be used in each step. For simplicity, it is assumed that the capacity of niches is equal to one. All agents are copied into a queue, which is then sorted in descending order, according to the objective fitness

value (lines 2–4). As long as the queue is not empty, the best remaining agent is removed from its head (line 8). Its effective fitness is set to be equal to the objective one. (line 9). Then, all other agents falling within a critical similarity radius are subject to clearing (lines 11–14). The effective fitness of the cleared agents is set to the minimum value from the original population (assuming a maximization problem), calculated in line 6. These cleared agents do not take part in further clearing, so they are removed from the queue.

The biggest advantage of this method with regard to EMAS is that it is fully transparent to the algorithm. All the basic actions and strategies are left unchanged. An inherent problem with the clearing procedure is the necessity to estimate the niche radius. However, this drawback can be eliminated by the use of hierarchical clustering [20], which could be done efficiently in an EMAS.

5. Preliminary experimental results

A detailed discussion of niching techniques performance is beyond the scope of this paper. However, the results of preliminary experimental investigations are presented below.

The two niching methods described in section 4, along with the basic EMAS from section 3, have been tested on a 10 and 100 dimensional Rastrigin function, a popular highly multimodal benchmark function for evolutionary algorithms. It has to be noted that the parameters of the niching techniques might not have been thoroughly optimized yet.

Two quantities were examined: a) the best fitness so far observed in the whole population up to a given step; and b) population diversity, calculated as its moment of inertia in the solution space [18].

It appears that the basic EMAS version currently outperforms the two considered niching techniques when it comes to fitness efficiency (Fig. 4). However, the gap is small for crowding in lower dimensions and for clearing in higher dimensions. Crowding in higher dimensions clearly does not converge.

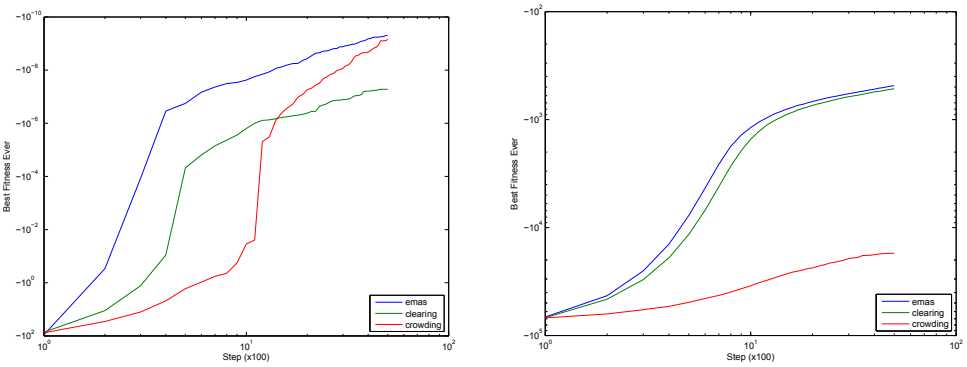
This lack of convergence of crowding in higher dimensions is presumably due to the fact that the population size is held constant. Therefore, the algorithm loses the adaptability inherent to EMAS, as is has been described in 4.1.

In the case of clearing , the smaller efficiency is most probably caused by the fact that it is difficult to estimate a good niche radius parameter [19]. This efficiency could be improved by meta-optimizing the parameter or even better, remove the need for such a parameter by introducing hierarchical clustering [20].

The counterweight for this decrease in efficiency is that, at least, a much higher diversity of the population is preserved, especially in lower dimensions (Fig. 5).

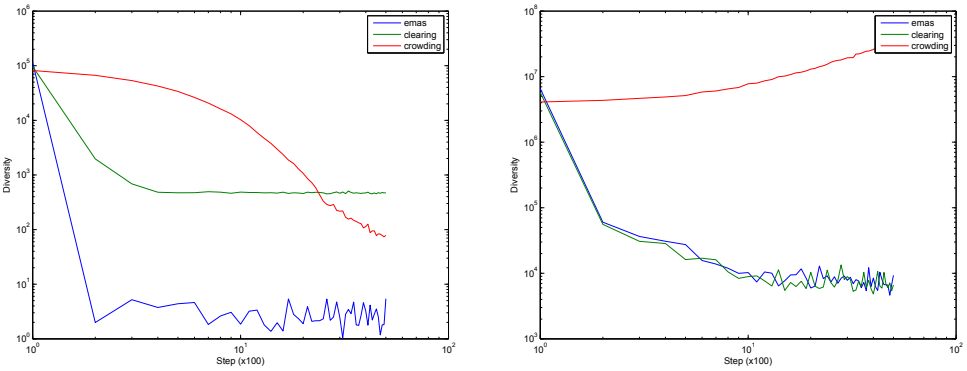
This results from the effective discovery of multiple optima, as it is shown in Fig. 6 – each dot represents an individual in the solution space. It can clearly be seen

that these do group around the regular local optima of the Rastrigin function and in that regard, clearing seams to be more efficient that crowding.



(a) 10 dimensional Rastrigin function (b) 100 dimensional Rastrigin function

Figure 4. Best fitness so far at a given step.

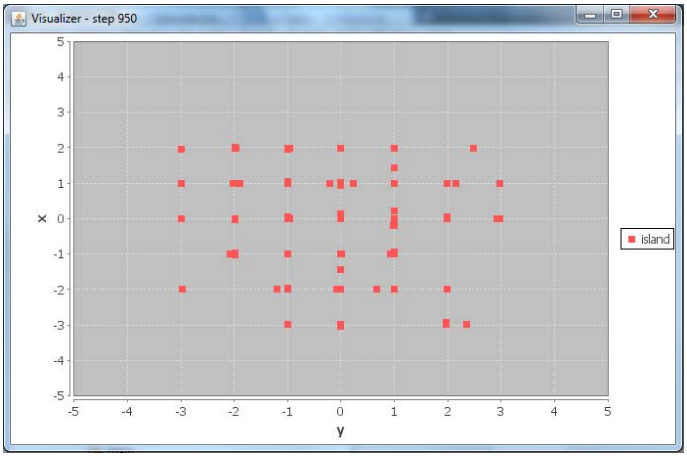


(a) 10 dimensional Rastrigin function (b) 100 dimensional Rastrigin function

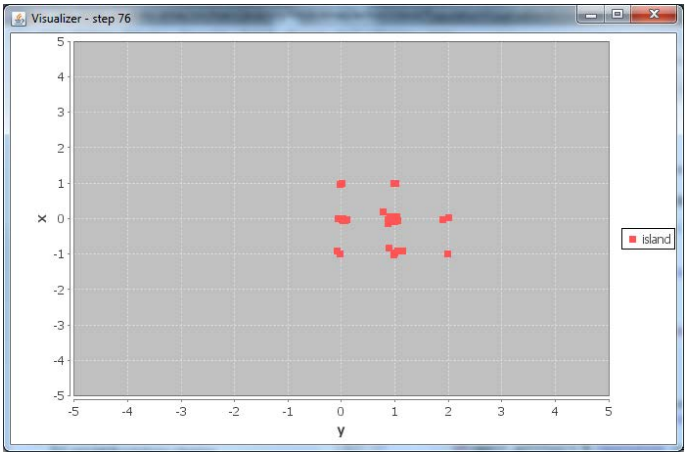
Figure 5. Population diversity at a given step.

6. Conclusion

Niching techniques in classical evolutionary algorithms attract great interest in the recent years. This results from the usefulness of these methods in solving several types of problems, for example multimodal or nonstationary optimization. Developing effective niching mechanisms for evolutionary multi-agent systems (EMAS) would allow to apply this class of computational methods to a much broader range of problems. It would also increase their usefulness in practical business problems, which by nature are most often computationally difficult, multimodal and nonstationary.



(a) Clearing procedure



(b) Crowding

Figure 6. Plot of the population in the solution space for a 2 dimentional Rastrigin function.

This papers proposes a flexible but simple algorithmical architecture for an EMAS. Further on, this paper demonstrates how to adapt existing classical niching methods to such an EMAS, on the example of two such methods: crowding and the clearing procedure. While crowding has assumptions incompatible with EMAS, the clearing procedure, however, might be used transparently to the rest of the algorithm.

The first experimental results indicate that the envisaged niching techniques preserve population diversity and have the population colonize multiple optima, which is especially valuable for multimodal optimization. However, as a counterweight, they do currently not outperform a raw EMAS when it comes to fitness efficiency. This

efficiency problem deserves further study and we hope to report new results on this matter in the future.

The conclusions arising from this work do not exhaust all the niching possibilities in evolutionary multi-agent systems. Here are some examples of possible further research areas:

- It would be interesting to adapt modern message-passing clustering algorithms, like affinity propagation [9], to an EMAS. As the agents population changes slowly, clustering could be performed in an efficient, incremental way. The results of such clustering could be leveraged in many niching techniques.
- According to the results described in [20], the introduction of hierarchical clustering increases the effectiveness of the clearing procedure and reduces its computational complexity. Again, hierarchical affinity propagation [10] could be used.
- Yet another possibility would be to combine the concept of a multinational evolutionary algorithm [22] with the idea of dynamic islands and agents flocks [13].

Instead of adapting existing, classical methods, one could also try to modify the basic algorithm of an EMAS to introduce niching in a dedicated way. A few interesting approaches could be:

- To modify the meeting strategy so that similar agents meet more often (like in crowding).
- To modify the energy transfer strategies so that similar agents take more energy from each other (like in the clearing procedure).
- To examine the impact of introducing topology into the environment. An appropriate topology, based on physical distance or similarity, could be used to make closer or similar agents to interact more often or in a stronger way.

Acknowledgements

I would like to express my gratitude to Marek Kisiel-Dorohinicki and Aleksander Byrski for their valuable advice and constructive suggestions, which helped me all along this research work.

The research presented here was partially supported by the grant "Biologically inspired mechanisms in planning and management of dynamic environments" funded by the Polish National Science Centre, No. N N516 500039.

References

[1] Byrski A., Kisiel-Dorohinicki M., Nawarecki E.: Agent-based evolution of neural network architecture. In *Applied Informatics*, ACTA Press, 2002.

[2] Byrski A., Schaefer R.: Stochastic model of evolutionary and immunological multi-agent systems: Mutually exclusive actions. *Fundamenta Informaticae*, 95(2):263–285, IOS Press, 2009.

[3] Cetnarowicz K., Kisiel-Dorohinicki M., Nawarecki E.: The application of evolution process in multi-agent world to the prediction system. In *Proc. of the Second*

International Conference on Multi-Agent Systems, ICMAS, vol. 96, pp. 26–32, 1996.

[4] De Jong K.: *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[5] Defaweux A., Lenaerts T., Maes S., Tuyls K., van Remortel P., Verbeeck K., Nowe A., Manderick B.: Niching and evolutionary transitions in mas. In *Proc. of GECCO 2001, presented at the Workshop on Evolutionary Computation and Multi-agent Systems, 7–11 July*, 2001.

[6] Dreżewski R.: A model of co-evolution in multi-agent system. *Multi-Agent Systems and Applications III*, pp. 1067–1067, Springer, 2003.

[7] Dreżewski R., Sepielak J., Siwik L.: Classical and agent-based evolutionary algorithms for investment strategies generation. *Natural Computing in Computational Finance*, pp. 181–205, Springer, 2009.

[8] Dreżewski R., Siwik L.: Co-evolutionary multi-agent system for portfolio optimization. *Natural Computing in Computational Finance*, pp. 271–299, Springer, 2008.

[9] Frey B., Dueck D.: Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[10] Givoni I., Chung C., Frey B.: Hierarchical affinity propagation. *Arxiv preprint arXiv:1202.3722*, 2012.

[11] Horn J.: *The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations*. PhD thesis, Citeseer, 1997.

[12] Jennings N.: On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, Elsevier, 2000.

[13] Kisiel-Dorohinicki M.: Flock-based architecture for distributed evolutionary algorithms. In *Artificial Intelligence and Soft Computing*, pp. 841–846. ICAISC, Springer, 2004.

[14] Mahfoud S.: Niching methods for genetic algorithms. *Urbana*, 51(95001), 1995. Citeseer.

[15] Mengshoel O., Galan S.: Generalized crowding for genetic algorithms. In *Genetic and Evolutionary Computation Conference 2010*, pp. 775–782. GECCO, 2010.

[16] Mengshoel O., Goldberg D.: Probabilistic crowding: Deterministic crowding with probabilistic replacement. In *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 409–416. GECCO, 1999.

[17] Mengshoel O., Goldberg D.: The crowding approach to niching in genetic algorithms. *Evolutionary Computation*, 16(3):315–354, 2008.

[18] Morrison R., De Jong K.: Measurement of population diversity. In *Artificial Evolution*, pp. 1047–1074. Springer, 2002.

[19] Petrowski A.: A clearing procedure as a niching method for genetic algorithms. In *Evolutionary Computation, 1996., Proc. of IEEE International Conference on*, pp. 798–803. IEEE, 1996.

[20] Pétrowski A.: An efficient hierarchical clustering technique for speciation. *Evolution. Technical report, Institute National des Telecommunications*, Evry, France, Technique Report, 1997.

[21] Schaefer R., Byrski A., Smołka M.: Stochastic model of evolutionary and immunological multi-agent systems: Parallel execution of local actions. *Fundamenta Informaticae*, 95(2):325–348, IOS Press, 2009.

[22] Ursem R. K.: Multinational evolutionary algorithms. In *Proc. of the Congress of Evolutionary Computation (CEC-99)*, vol. 3, pp. 1633–1640, 1999.

Affiliations

Daniel Krzywicki
AGH University of Science and Technology, Krakow, Poland, `krzywic@agh.edu.pl`

Received: 5.09.2012
Revised: 28.09.2012
Accepted: 3.12.2012