An On-line On-board Distributed Algorithm for Evolutionary Robotics

Robert-Jan Huijsman, Evert Haasdijk, and A.E. Eiben

Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
r.huijsman@student.vu.nl, e.haasdijk@vu.nl, a.e.eiben@vu.nl
http://www.cs.vu.nl/ci/

Abstract. Imagine autonomous, self-sufficient robot collectives that can adapt their controllers autonomously and self-sufficiently to learn to cope with situations unforeseen by their designers. As one step towards the realisation of this vision, we investigate on-board evolutionary algorithms that allow robot controllers to adapt without any outside supervision and while the robots perform their proper tasks. We propose an EVAG-based on-board evolutionary algorithm, where controllers are exchanged among robots that evolve simultaneously. We compare it with the $(\mu + 1)$ ON-LINE algorithm, which implements evolutionary adaptation inside a single robot. We perform simulation experiments to investigate algorithm performance and use parameter tuning to evaluate the algorithms at their *best possible* parameter settings. We find that distributed on-line on-board evolutionary algorithms that share genomes among robots such as our EVAG implementation effectively harness the pooled learning capabilities, with an increasing benefit over encapsulated approaches as the number of participating robots grows.

Keywords: evolutionary robotics, on-line evolution, distributed evolution

1 Introduction

The work presented in this paper is inspired by a vision of autonomous, self-sufficient robots and robot collectives that can cope with situations unforeseen by their designers. An essential capability of such robots is the ability to adapt –evolve, in our case– their controllers in the face of challenges they encounter in a hands-free manner, "the ability to learn control without human supervision," as Nelson *et al.* put it [13]. In a scenario where the designers cannot predict the operational circumstances of the robots (e.g, an unknown environment or one with complex dynamics), the robots need to be deployed with roughly optimised controllers and the ability to evolve their controllers autonomously, on-line and on-board.

This contrasts with the majority of evolutionary robotics research, which focusses on *off-line* evolution, where robot controllers are developed -evolved- in a separate training stage before they are deployed to tackle their tasks in earnest.

When dealing with multiple autonomous robots, one can distinguish three options to implement on-line evolution [7]:

Encapsulated Each robot carries an isolated and self-sufficient evolutionary algorithm, maintaining a population of genotypes inside itself;

Distributed Each robot carries a single genome and the evolutionary process takes place by exchanging genetic material between robots;

Hybrid Which combines the above two approaches: each robot carries multiple genomes and shares these with its peers.

In this paper, we compare instances of each of these three schemes: the encapsulated $(\mu + 1)$ ON-LINE algorithm, the distributed Evolutionary Agents algorithm (EVAG) [9] and a hybrid extension of EVAG.

One of EVAG's distinguishing features is that it employs the newscast algorithm [8] to exchange genomes between peers (in our case: robots) and to maintain an overlay network for peer-to-peer (robot-to-robot) communication. Newscast-based EVAG has proved very effective for networks of hundreds or even thousands of peers, but in the case of swarm robotics, network sizes are likely to be much smaller. Therefore, it makes sense to compare the efficacy of newscast-based EVAG with a panmictic variant where the overlay network is fully connected.

It is well known that the performance of evolutionary algorithms to a large extent depends on their parameter values [11]. To evaluate the algorithms at their *best possible* parameter setting, we tune the algorithm parameters with REVAC, an evolutionary tuning algorithm specifically designed for use with evolutionary algorithms [10].

Summarising, the main question we address in this paper is: how do the three algorithms compare in terms of performance and can we identify circumstances in which to prefer one of the three schemes over the others? Secondly, we investigate how EVAG's newscast population structure influences its performance compared to a panmictic population structure. Thirdly, we briefly consider the sensitivity of the algorithms to parameter settings.

2 Related work

The concept of on-board, on-line algorithms for evolutionary robotics was discussed as early as 1995 in [15], with later research focussed on the 'life-long learning' properties of such a system [20].

A distributed approach to on-line, on-board algorithms was first investigated as 'embodied evolution' in [21], where robots exchange single genes at a rate proportional to their fitness with other robots that evolve in parallel. Other work on on-line evolution of robot controllers is presented in [4] that describes the evolution of controllers for activating hard-coded behaviours for feeding and mating. In [1], Bianco and Nolfi experiment with open-ended evolution for robot swarms with self-assembling capabilities and report results indicating successful evolution of survival methods and the emergence of multi-robot individuals with co-ordinated movement and co-adapted body shapes.

The $(\mu + 1)$ ON-LINE algorithm – this paper's exemplar for the encapsulated approach – has been extensively described in [7] and [2], where it was shown to be capable of evolving controllers for a number of tasks such as obstacle avoidance, phototaxis and patrolling. [5] uses encapsulated evolution to evolve spiking circuits for a fast forward task. Encapsulated on-line evolution as a means for continuous adaptation by using genetic programming is suggested in [16].

The distributed approach to on-line evolutionary robotics has a clear analogy with the field of parallel evolutionary algorithms, in particular to the fine-grained approach, where each individual in the population has a processor of its own. The primary distinguishing factor among fine-grained parallel algorithms is their population structure, with small-world graphs proving competitive with panmictic layouts [6].

The hybrid scheme can be implemented as what in parallel evolutionary algorithms is known as the island model: the population is split into several separately evolving sub-populations (the islands), that occasionally exchange genomes. This approach is used in [19] and [4]. A variant where the robots share a common hall of fame is implemented in [12]. As will become apparent, we take a slightly different approach where genome exchange between sub-populations is the norm.

3 Algorithms

Autonomous on-line adaptation poses a number of requirements that regular evolutionary algorithms don't necessarily have to contend with. We take a closer look at two especially relevant considerations.

To begin with, fitness must be evaluated *in vivo*, i.e., the quality of any given controller must be determined by actually using that controller in a robot as it goes about its tasks. Such real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances; for instance, the nth controller will start at the final location of the $(n-1)$th one. This leads to very dissimilar evaluation conditions and ultimately to very noisy fitness evaluations. To address this issue, the algorithms we investigate here implement re-evaluation: whenever a new evaluation period commences, the robot can choose (with a probability ρ) not to generate a new individual but instead re-evaluate an existing individual to refine the fitness assessment and so combat noise.

The second issue specific to on-line evolution is that, in contrast to typical applications of evolutionary algorithms, the best performing individual is not the most important factor when applying on-line adaptation. Remember that controllers evolve as the robots go about their tasks; if a robot continually evaluates poor controllers, that robot's *actual* performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display a more or less stable but improving level of performance throughout the continuing search.

3.1 $(\mu + 1)$ ON-LINE

The $(\mu + 1)$ ON-LINE algorithm is based on the classical $(\mu + 1)$ evolutionary strategy [17] with modifications to handle noisy fitness evaluations and promote rapid convergence. It maintains a population of μ individuals within each robot and these are evaluated in a time-sharing scheme, using an individual's phenotype as the robot's controller for a specified number of time units. A much more detailed description of $(\mu+1)$ ON-LINE is given in [7].

3.2 EVAG

EVAG was originally presented in [9] as a peer-to-peer evolutionary algorithm for parallel tackling of computationally expensive problems, with the ability to harness a large number of processors effectively. The analogies between parallel evolutionary algorithms and a swarm of robots adapting to their environment and tasks in parallel make EVAG a suitable candidate for an on-board, on-line distributed evolutionary algorithm for evolutionary robotics.

The basic structure of EVAG is straightforward and similar to a $1 + 1$ evolution strategy: each peer (robot) maintains a record of the best solution evaluated by that peer up until that point – the champion. For every new evaluation a new candidate is generated, using crossover and mutation; if the candidate outperforms the current champion it replaces the champion.

The basic definition of EVAG leaves many decisions open to the implementer, such as the choice of recombination and mutation operators and the details of parent selection (other than that it should select from peers' champions). Because we are interested in the effects of the distributed nature of the algorithm rather than those due to, say, different recombination schemes, we have chosen our evolutionary operators to match the $(\mu + 1)$ ON-LINE algorithm. As a result, the only difference between EVAG and $(\mu + 1)$ ON-LINE (with $\mu = 1$) lies in the exchange of genomes between robots and using a cache of received genomes rather than only a locally maintained population when selecting parents.

In this light, the extension of regular EVAG to a hybrid form is a straightforward one: rather than maintaining only a single champion on-board, the robots now maintain a population of μ individuals locally. With $\mu = 1$, this implements the distributed scheme, with $\mu > 1$, it becomes a hybrid implementation. With the cache of received genomes disabled, it boils down to $(\mu + 1)$ ON-LINE, our encapsulated algorithm. The pseudo code in algorithm 1 illustrates the overlap between these three implementations.

EVAG normally uses newscast [8] to exchange solutions efficiently while maintaining a low number of links between peers: each robot locally maintains a cache of recently received genomes. Periodically, each robot randomly takes one of the genomes in its cache and contacts the robot at which that genome originated. These two robots then exchange the contents of their cache. When needed, parents are selected from the union of this cache and the local champion using binary tournament selection. Because [8] showed that with this update scheme, picking a genome randomly from the cache of received genomes is all but equivalent to picking one randomly from the entire population, this assures that the binary tournament takes place as if the contestants were randomly drawn from the combined population across all robots.

Earlier research showed very promising results for EVAG [9], but these results were obtained using thousands of nodes, while evolutionary robotics generally takes place with group sizes of no more than a few dozen robots. To investigate if EVAG's newscast overlay network remains efficient in these smaller populations we evaluate not only the

```
for $i \leftarrow 1$ to $\mu$ do                                  // Initialisation
    population[i] ← CreateRandomGenome();
    population[i].$\sigma \leftarrow \sigma_{initial}$; // Mutation step size, updated cf. [2]
    population[i].Fitness ← RunAndEvaluate(population[i]);
end
for ever do                                              // Continuous adaptation
    if random() $< \rho$ then // Don't create offspring, but re-evaluate
    selected individual
        Evaluatee ← BinaryTournament(population);
        Evaluatee.Fitness ← (Evaluatee.Fitness + RunAndEvaluate(Evaluatee)) / 2;
        // Combine re-evaluation results through exponential
        moving average
    else    // Create offspring and evaluate that as challenger
        ParentA ← BinaryTournament(pool of possible parents);
        ParentB ← BinaryTournament(pool of possible parents - parentA);
        if random() $< crossoverRate$ then
            Challenger ← AveragingCrossover(ParentA, ParentB);
        else
            Challenger ← ParentA;
        end
        if random() $< mutationRate$ then
            Mutate(Challenger);         // Gaussian mutation from $N(0,\sigma)$
        end
        Challenger.Fitness ← RunAndEvaluate(Challenger);
        if Challenger.Fitness $>$ population[$\mu$].Fitness then      // Replace last
        (i.e. worst) individual in population w. elitism
            population[$\mu$] ← Challenger;
            population[$\mu$].Fitness ← Challenger.Fitness;
        end
    end
    Sort(population);
end
```

Algorithm 1: The on-line evolutionary algorithm. For $(\mu + 1)$ ON-LINE, the pool of possible parents is the on-board population of size μ; for both regular and hybrid EVAG, it is the union of individuals received from the robot's peers through newscast and the on-board population (with $\mu = 1$ for standard EVAG).

standard newscast-based EVAG, but also an EVAG variant that uses a panmictic population structure where a robot's choice of genomes for the binary tournament parent selection is truly uniform random from the entire population across all robots. This variant of EVAG requires full connectivity among peers (robots) and is therefore not suitable for use in truly large-scale applications. However, evaluation of the panmictic structure compared to that of newscast is interesting, since it allows us to determine the performance penalty of using of a peer-to-peer approach.

4 Experiments

To investigate the performance of the algorithms and their variants we conduct experiments with simulated e-pucks in the ROBOROBO[1] environment. The experiments have the robots running their own autonomous instance of $(\mu + 1)$ ON-LINE, EVAG or its hybrid extension EVAG, governing the weights of a straightforward perceptron neural net controller with hyperbolic tangent activation function. The neural net has 9 input nodes (8 sensors and a bias), no hidden nodes and 2 output nodes (the left and right motor values for the differential drive), giving a total of 18 weights. To evolve these 18 weights, the evolutionary algorithm uses the obvious representation of real-valued vectors of length 18 for the genomes.

The robots' task is movement with obstacle avoidance: they have to learn to move around in a constrained arena with numerous obstacles as fast as possible while avoiding the obstacles. The robots are positioned in an arena with a small loop and varied non-looping corridors (see Fig. 1). The fitness function we use has been adapted from [14]; it favours robots that are fast and go straight ahead. Fitness is calculated as follows:

$$f = \sum_{t=0}^{\tau} (v_t \cdot (1 - v_r)) \qquad (1)$$

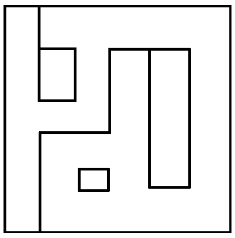


Fig. 1. The arena

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation). In our simulations, whenever a robot touches an obstacle, $v_t = 0$, so the fitness increment for time-steps where the robot is in collision is 0. A good controller will turn only when necessary to avoid collisions and try to find paths that allow it to run in a straight line for as long as possible.

We run our simulations with each robot in an arena of its own so that they can't get in each others' way, although the robots obviously can communicate across arena instances. The reasons for this are twofold: firstly, eliminating physical interaction between the robots ensures that a robot's performance is due to its own actions rather than that of others around it; this allows us a clearer view of the effects of genome exchange. Secondly, it allows us to scale our simulations from a very small number of robots to a very large number of robots while using the exact same arenas; this guarantees that in those cases any change in performance of the robots is due to their increased group size rather than a change in environment.

We evaluate the EVAG variants with group sizes of 4, 16, 36 and 400 robots: we hypothesise that differences in group size influence the performance of distributed and hybrid algorithms due to their facilities for genome exchange. Since a larger group of robots is able to evaluate a larger number of candidate solutions simultaneously, the odds of finding a successful genome are higher. EVAG is intended to distribute these successful genomes across all robots, thus improving the performance of the entire group.

[1] `http://www.lri.fr/~bredeche/roborobo/`

Because the experiments were designed so that genome exchange is the only possible interaction between robots in a group, the robots can have no physical interaction. Without physical interaction and no genome exchange the *average* performance of a group of 4 robots running $(\mu + 1)$ ON-LINE would be identical to that of a group of 400 robots doing the same. Since average performance is our only metric for success and since for $(\mu + 1)$ ON-LINE this metric is not influenced by the number of robots in the experiment, we can perform experiments for $(\mu + 1)$ ON-LINE for a group of 4 robots and use these results as a fair comparison to an experiment with EVAG using 400 robots. We can therefore safely omit the costly $(\mu+1)$ ON-LINE simulations for the group sizes 16, 36 and 400 as their performance would be the same as that of the group of 4.[2]

Rather than comparing the algorithms at identical (so far as possible) parameter settings, we compare the performance at their relative *best possible* parameter settings after tuning as described in Sec 4.1. Note that, where applicable, values of μ may vary from one experiment to the next: this does not imply that the number of evaluations is influenced as the robots have a fixed amount of (simulated) wall-clock time in which to learn the task.

4.1 Evaluation with parameter tuning

It is a well-known fact that the performance of an evolutionary algorithm is greatly determined by its parameter values. Despite this, many publications in the field of evolutionary computing evaluate their algorithms using fairly arbitrary parameter settings, based on ad hoc choices, conventions, or a limited comparison of different parameter values. This approach can easily lead to misleading comparisons, where method A is tested with very good settings and method B based on poor ones. An recommendable alternative is the use of *automated parameter tuning*, where a tuning algorithm is used to optimize the parameter values and one compares the best variants of the evolutionary algorithms in question [3]. This approach helps prevent misleading comparions.

In our experiments with 4, 16 and 36 robots we evaluate the performance of the algorithms by performing parameter tuning for a fixed length of time and comparing the results. We use the MOBAT toolkit[3] to automate our tuning process. MOBAT is based on REVAC [10], which has been shown to be an efficient algorithm for parameter tuning [18]. For every combination of robot group size and algorithm MOBAT evaluates 400 parameter settings; each parameter setting is tested 25 times to allow statistically significant comparisons. Unless otherwise specified, performance comparisons were made with the best-performing parameter setting that was found for each of the algorithms-group size combinations.

Due to the computational requirements of tuning the parameters of very large simulations it was infeasible to tune parameters for our experiments with a group size of 400 robots. Instead, these experiments were performed at the parameter settings found for the 36 robots.

Each algorithm variant has its own parameters that need tuning. These parameters and the range within which they were tuned are listed in Table 1.

[2] Source code and scripts to repeat our experiments can be found at `http://www.few.vu.nl/~ehaasdi/papers/EA-2011-EvAg`.

[3] `http://sourceforge.net/projects/mobat/`

Parameter	*Tunable range*
τ (Evaluation time steps per candidate)	300 – 600
μ (Population size – for encapsulated and hybrid schemes)	3 – 15
$\sigma_{initial}$ (Initial mutation step size)	0.1 – 10.0
ρ (Re-evaluation rate)	0.0 – 1.0
Crossover rate	0.0 – 1.0
Mutation rate	0.0 – 1.0
Newscast item TTL (for newscast-based EVAG variants)	3 – 20
Newscast cache size (for newscast-based EVAG variants)	2 – group size

Table 1. The parameters as tuned by REVAC.

5 Results and Discussion

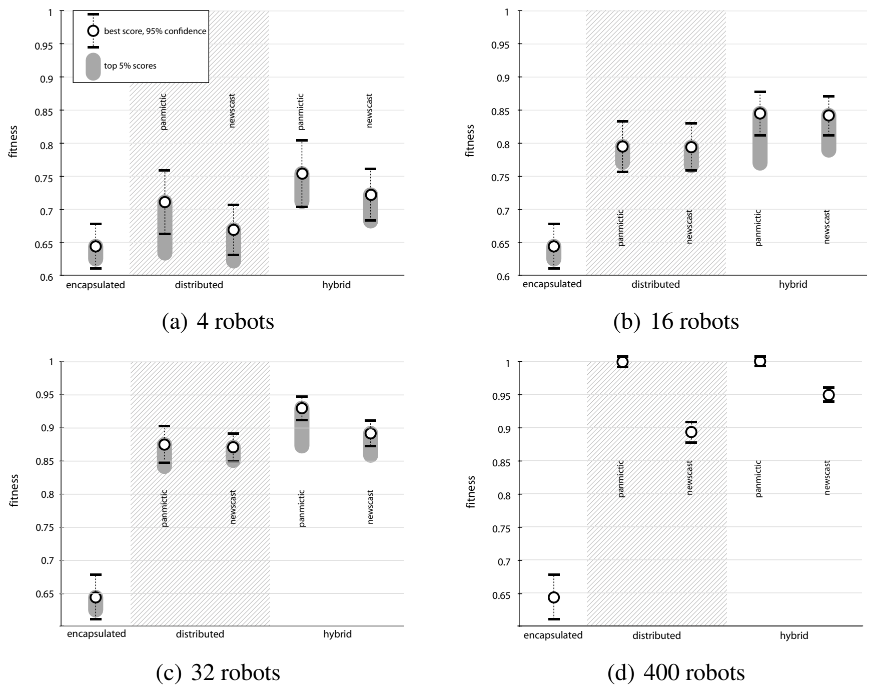
Fig. 2 shows the results for the experiments for group sizes 4, 16, 36 and 400. Each graph shows the results for $(\mu + 1)$ ON-LINE (*encapsulated*), for EVAG (*distributed*) and for EVAG's *hybrid* extension. The latter two have results for the panmictic as well as the newscast variant. The white circles indicate the average performance (over 25 repeats) of the best parameter vector for that particular algorithm variant and the whiskers extend to the 95% confidence interval using a t-test. To investigate how sensitive the algorithm variants are to the choice of parameter settings, we also show the performance variation in the top 5% of parameter vectors, indicated by the grey ovals. The results are normalised so that the highest performance attained overall is 1. Note that because –as discussed above– we only ran $(\mu + 1)$ ON-LINE experiments with group size 4, the data for $(\mu + 1)$ ON-LINE are the same in all four graphs.

The performances shown are always the average performance of the entire group of robots in an experiment, not just the performance of the best robot: we are interested in developing an algorithm that performs well for *all* robots, rather than an algorithm that has a very high peak performance in one robot but does not succeed in attaining good performance for the entire group. Performances are compared using a t-test with $\alpha = 0.05$.

In the scenario with four robots (Fig. 2(a)) the hybrid algorithm significantly outperforms the encapsulated scheme, but it is not significantly better than the distributed scheme. The differences between the distributed and the encapsulated scheme are not significant. The same goes for the differences between the panmictic and the newscast variants. All four EVAG variants show a relatively large variation in the top-5% performances, indicating that they are more sensitive to the quality of parameter settings than is $(\mu + 1)$ ON-LINE.

It is surprising that the distributed scheme matches (even improves, although not significantly) performance with the encapsulated scheme when we consider that four robots running EVAG have access to only four (shared) genotypes, compared to the 12 (isolated) genotypes that are stored by each of the robots running $(\mu+1)$ ON-LINE (with μ tuned to 12).

With group size 16 (Fig. 2(b)), EVAG starts to come into its own: both the distributed and the hybrid schemes outperform $(\mu + 1)$ ON-LINE significantly, although the differences among the EVAG variants are not significant at 95%. Moreover, the distributed variants now perform almost as consistently as their encapsulated counterpart:



(a) 4 robots

(b) 16 robots

(c) 32 robots

(d) 400 robots

Fig. 2. Performance plots for various group sizes: performance is averaged over 25 repeats. Note that there was no tuning for group size 400 and hence no top 5% parameter vectors.

both variation in scores for a single parameter setting and the variation of scores in the top-5% are at the level of $(\mu + 1)$ ON-LINE, indicating that the distributed algorithm becomes less sensitive to sub-optimal parameter settings as the number of robots participating in the evolution grows.

With the increase in group size from 16 to 36 (Fig. 2(c)) EVAG again shows a significant performance increase and the confidence intervals and range of top-5% values are further reduced. For the first time we see a significant difference between the EVAG variants, with the panmictic hybrid approach performing significantly better than the alternatives.

Finally, Fig. 2(d) shows the results for 400 robots. The computational requirements of so large a simulation make parameter tuning infeasible; instead we investigate the performance of the algorithms at the best parameter settings found for the 36-robot scenario. The plot shows a large jump in performance for the panmictic variants, a respectable increase for the hybrid newscast variant, but little difference for the distributed newscast implementation. For all EVAG variants the confidence interval of the scores has shrunk considerably, indicating that EVAG continues to become more reliable as the robot group size increases.

The results show that, as the group size increases, there is an increasing benefit to using an algorithm such as (the hybrid extension of) EVAG rather than a purely encapsulated algorithm such as $(\mu + 1)$ ON-LINE. In particular, the hybrid scheme consistently reaches the highest performance (although the difference with the distributed scheme is rarely shown to be significant at 95%). EVAG's panmictic variants perform consistently better than its newscast-based version, but the difference is only significant at the largest group sizes we consider. In truly large-scale environments it would be infeasible to have a panmictic population structure; unfortunately it is especially in the large-scale 400-robot scenario that the newscast-based EVAG lags behind its panmictic counterpart.

One possible explanation for the lower performance of the newscast-based EVAG is that it has two extra parameters to tune (newscast cache size and the news items' TTL), making it more difficult for REVAC to find optimal parameter settings within the 400 attempts that we allowed it. However, this does not explain the large performance gap in the 400-robot scenario, where the same parameter settings as in the 36-robot scenario were used. One suspect is the 'newscast cache size' parameter, of which an interesting trend can be seen when looking at the progression of the value across the different scenarios: as the number of robots grows, so does the value of the cache size parameter. Earlier research on newscast suggests that a cache size of 10 should be sufficient for very large-scale applications [8], but nevertheless REVAC favours higher numbers, approximately in the range of $\frac{3}{4}$th of the number of robots. To see if the setting of a cache size of 27 is sub-optimal for the 400-robot scenario we have investigated if increasing the cache size to 300 leads to an improved performance; this turned out not to be the case, with both cache sizes performing at the same level. This indicates that a lack of cache space is not to blame for the gap in performance between newscast-based EVAG and its panmictic counterpart and its exact cause remains unknown.

6 Conclusion

In this paper we have compared the $(\mu + 1)$ ON-LINE on-board encapsulated algorithm to a distributed and a hybrid implementation of EVAG for on-line, on-board evolution of robot controllers. We have performed simulation experiments to investigate the performance of these algorithms, using automated parameter tuning to evaluate each algorithm at its *best possible* parameter setting.

Comparing the algorithms in terms of performance, EVAG performs consistently better than $(\mu + 1)$ ON-LINE, with the effect being especially prominent when the number of robots participating in the scenario is large. Even at the smallest group size we considered, the distributed scheme is competitive, despite having a population of only four individuals.

To estimate the sensitivity to sub-optimal parameter settings we have observed the variation in performance in the top-5% of parameter vectors. We have seen that $(\mu + 1)$ ON-LINE is quite stable, with the top-5% close to the best performance. In both the distributed and the hybrid variant, EVAG's performance is quite unstable when the group of robots is small, but becomes increasingly reliable as the group size increases.

For large numbers of robots, the newscast population structure has a small negative effect on the performance of the EVAG variants when compared to a panmictic population structure. However, in a large-scale scenario it may be infeasible to maintain a panmictic population structure. In those scenarios the small performance loss when using a newscast-based population structure may well be outweighed by the practical advantages of being able to implement the algorithm at all.

Although we have only performed experiments with a single and quite straightforward task, we conclude that both the distributed and hybrid approaches to on-board on-line evolutionary algorithms in evolutionary robotics are feasible and provide a promising direction for research in this field.

The hybrid scheme can be preferred over the encapsulated and the distributed case because it efficiently harnesses the opportunities of parallelising the adaptation process over multiple robots while performing well even for small numbers of robots. Although the panmictic variant does outperform the newscast-based implementation for very large numbers of robots, we do not know if or how tuning specifically for 400 robots would have influenced the apparent performance difference between newscast and panmixia. We would still prefer the latter because of its inherent scalability and robustness.

Future research should confirm our findings in different scenarios. Additionally, there is research to be done regarding the study of which parameters are influential and why certain parameter settings are more effective than others.

Acknowledgements This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the SYMBRION project under grant agreement 216342. The authors would like to thank Selmar Smit and our partners in the SYMBRION consortium for many inspirational discussions on the topics presented here.

References

1. Raffaele Bianco and Stefano Nolfi. Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 4:227–248, 2004.
2. A. E. Eiben, Giorgos Karafotias, and Evert Haasdijk. Self-adaptive mutation in on-line, on-board evolutionary robotics. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 147–152. IEEE Press, Piscataway, NJ, September 2010.
3. A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011. to appear.
4. S. Elfwing, E. Uchibe, K. Doya, and H.I. Christensen. Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK, 2-5 September 2005. IEEE Press.
5. Dario Floreano, Nicolas Schoeni, Gilles Caprari, and Jesper Blynel. Evolutionary bits'n'spikes. In Russell K. Standish, Mark A. Bedau, and Hussein A. Abbass, editors, *Artificial Life VIII : Proceedings of the eighth International Conference on Artificial Life*, pages 335–344, Cambridge, MA, USA, 2002. MIT Press.
6. Mario Giacobini, Mike Preuss, and Marco Tomassini. Effects of scale-free and small-world topologies on binary coded self-adaptive CEA. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP˜2006*, volume 3906 of *LNCS*, pages 85–96, Budapest, April 2006. Springer Verlag.

7. Evert Haasdijk, A. E. Eiben, and Giorgos Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010. IEEE Computational Intelligence Society, IEEE Press.
8. Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the internet. Technical report, Vrije Universiteit Amsterdam, 2002.
9. J. L. Laredo, A. E. Eiben, M. van Steen, and J. J. Merelo. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11:227–246, June 2010.
10. V Nannen and A E Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. *Proc of IJCAI 2007*, pages 975–980, 2007.
11. Volker Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pages 528–538, Berlin, Heidelberg, 2008. Springer-Verlag.
12. Ulrich Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In C.G. Prince, Y. Demiris, Y. Marom, H. Kozima, and C. Balkenius, editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK, August 2002. LUCS.
13. Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.
14. Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.
15. Peter Nordin and Wolfgang Banzhaf. Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67. AAAI, 1995.
16. Peter Nordin and Wolfgang Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5:107–140, 1997.
17. Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.
18. S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 399–406, Piscataway, NJ, USA, 2009. IEEE Press.
19. Yukiya Usui and Takaya Arita. Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215, 2003.
20. Joanne H. Walker, Simon M. Garrett, and Myra S. Wilson. The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(2):423–432, 2006.
21. Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1 – 18, 2002.