Asynchronous Actor-based Approach to Multiobjective Hierarchical Strategy *

Michał Idzik[0000−0002−8446−4966], Aleksander Byrski[0000−0001−6317−7012], Wojciech Turek, and Marek Kisiel-Dorohinicki[0000−0002−8459−1877]

AGH University of Science and Technology
{miidzik,olekb,wojciech.turek,doroh}@agh.edu.pl

Abstract. Hierarchical Genetic Strategy (HGS) is a general-purpose optimization metaheuristic based on multi-deme evolutionary-like optimization, while demes are parts of adaptive dynamically changing tree. The paper focuses on adaptation of the classic HGS algorithm for multi-criteria optimization problems, coupling the HGS with Particle Swarm Optimization demes. The main contribution of the paper is showing the efficacy and efficiency of the actor-based implementation of this metaheuristic algorithm.

Keywords: hierarchic genetic search · metaheuristics · actor model.

1 Introduction

Keeping balance between exploration and exploitation is a crucial task of an expert, trying to solve complex problems with metaheuristics. A number of different approaches were made (like automatic adaptation of variation operators in evolution strategies [1] or multi-deme evolution model [2].

About 20 years ago an interesting metaheuristics has been proposed by Schaefer and Kolodziej, namely Hierarchic Genetic Search [12], managing a whole tree of demes which can be dynamically constructed or removed, depending on the quality of their findings. Recently this metaheuristic was adapted to solving multi-criteria optimization problems by Idzik et al. [9], and coupled with Particle Swarm Optimization as working nodes (demes) instead of classic genetic algorithms. This new algorithm has proven to be efficient in many multi-criteria benchmark problems, however as it is quite normal in the case of metaheuristics, it is cursed with high computational complexity.

Nowadays HPC-related solutions (supercomputers, hybrid infrastructures etc.) are very common and easy to access, especially for scientists[1]. High-level programming languages, such as Scala/Akka or Erlang make the use of such facilities

* The research presented in this paper was financed by Polish National Science Centre PRELUDIUM project no. 2017/25/N/ST6/02841.

[1] E.g. Academic Computing Centre "Cyfronet" of AGH University of Science and Technology makes possible to utilize supercomputing facilities to all scientists working in Poland or cooperating with Polish scientists, free of charge.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

very easy, and significantly decrease steepness of the learning curve for people who want to try such solutions.

This paper deals with a concept of actor-based design and implementation of Multiobjective HGS aiming at showing that such an approach yields a very efficient and efficacious computing system. The next section discusses existing actor-based implementations of metaheuristics, then the basics of HGS are given, later the actor model of HGS is presented and the obtained experimental results are shown and discussed in detail. Finally the paper is concluded and future work plans are sketched out.

2 Actor-based implementation of metaheuristics

Actor-based concurrency is a popular, easy to apply paradigm for parallelization and distribution of computing. As this paper is focused on parallelization of a complex metaheuristic algorithm using the actor-based approach, let us refer to existing similar high-level distributed and parallel implementations of metaheuristics.

Evolutionary multi-agent system is an interesting metaheuristic algorithm putting together the evolutionary and agent-based paradigms. Thus an agent becomes not only a driver for realization of certain computing task, but also a part of the computation, carrying the solution (genotype) and working towards improving its quality throughout the whole population. The agents undergo decentralized selection process based on non-renewable resources assigned to agents and the actions of reproduction and death. Such an approach (high decentralization of control) resulted in several distributed and parallel implementations, while one utilizing actor-model and defining so-called "arenas" was particularly efficient and interesting.

Its first implementation was realized in Scala [8]. The arenas were designed as meeting places, where particular actors (agents) can go and do a relevant task. E.g. at the meeting arena the agents were able to compare the qualities of their solutions while at the reproduction arena they could produce offspring etc. Another very efficient implementation of EMAS based on actor model was realized using Erlang [18]. Those implementations were tested in supercomputing environment and yielded very promising results.

Another interesting high-level approach to parallelization of metaheuristics consists in using a high-level parallel patterns in order to be able to automatically parallelize and distribute certain parts of code [17]. This approach has also been widely tested on many available supercomputing facilities.

Finally, a high-level distributed implementation of Ant Colony Optimization type algorithms was realized using Scala and Akka [16]. The approach is based on distributing the pheromone table and assuring the updates of its state. Currently the research shows that it might be possible to accept certain delays or even lack of updates of the pheromone table, still maintaining the high quality and scalability of the computing.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

Observing efficient outcomes of the above-mentioned approaches, we decided to apply actor-based concurrency model for implementing HGS.

3 Hierarchic Genetic Strategy

Another approach involving multiple evolving populations is Hierarchic Genetic Strategy (HGS) [12]. The algorithm dynamically creates subpopulations (*demes*) and lays them out in tree-like hierarchy (see Fig. 1). The search accuracy of a particular deme depends on its depth in the hierarchy. Nodes closer to the root perform more chaotic search to find promising areas. Each tree node is assigned with an internal evolutionary algorithm (*driver*, in single-objective processing it is the Simple Genetic Algorithm).

The process of HGS can be divided into several steps, called *metaepochs*. Each metaepoch consists of several epochs of a driver. Additionally, HGS-specific mechanics are applied. The most promising individuals of each node have a chance to become seeds of next-level *child nodes* (*sprouting* procedure). A child node runs with reduced variance settings, so that its population will mostly explore that region. To eliminate risk of redundant exploration of independently evolving demes, branch comparison procedure is performed. If the area is already explored by one of other children, then sprouting is cancelled and next candidate for sprouting is considered. Furthermore, the *branch reduction* compares and removes populations at the same level of the HGS tree that perform search in the common landscape region or in already explored regions.

Following very good results obtained for single-objectve problems, the HGS metaheuristic has been adapted for multi-objectve optimization tasks (MO-HGS [3]). This direction was further explored and improved [9] to create more generic solution, able to incorporate any multi-objective evolutionary algorithm (MOEA) as HGS driver.

The basic structure of this approach, denoted as Multiobjective Optimization Hierarchic Genetic Strategy with maturing (MO-mHGS) is similar to classical HGS, but the following features of the basic algorithm had to be adapted in order to tackle multi-objective problems:

- Flexibility of proposed model – different MOEA approaches may vary substantially. There are algorithms basing on Pareto dominance relation, algorithms built around quality indicator or algorithms decomposing multi-objective problem into subproblems to handle larger set of objectives. Moreover, some of them are generational approaches, others are steady-state. They all should be supported as MO-mHGS drivers.
- Evaluating quality of internal algorithm outcome – MOEA have specific set of quality indicators measuring distance from Pareto front, individuals spread, coverage of separate parts of a front, etc.
- Killing of the node based on lack of improvement of the solution in the last metaepoch – in MO-mHGS hypervolume metric is used as a factor for checking the spread of the population processed at the node.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

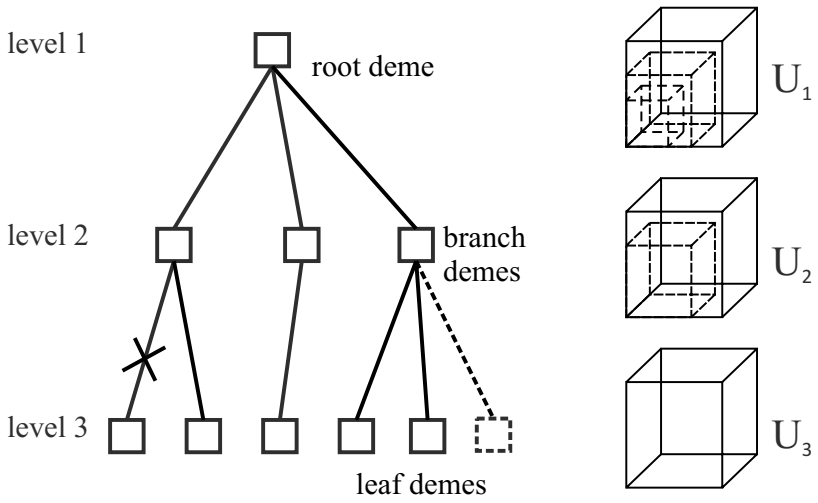


Fig. 1. Left panel: exemplary HGS tree with three levels. The left-most leaf is being reduced, the right-most leaf is being created. Right panel: three-dimensional genetic spaces in the real-number encoding, corresponding to levels in HGS tree.

Hierarchic strategies are also a subject of hybridization. They can be combined with local methods, e.g. local gradient-based search [6], and with clustering ([11] and [7], for HGS and MO-HGS, respectively).

4 Actor-based HGS models

In recent research we have shown [9] that hierarchical multi-deme model can be used in combination with any single-deme MOEA. We considered mainly variable evaluation accuracy as a crucial MO-mHGS feature. The results were promising and proved its applicability to real-world problems, where time of calculating a single fitness evaluation depends on the accuracy.

However, HGS model has additional properties we can take advantage of. Similarly to other multi-deme aproaches such as Island Model [10], calculations can be naturally divided into several processes. These processes are able to run simultaneously, ensuring benefits of parallel execution.

4.1 Adapting HGS to asynchronous environment

In classic MO-mHGS initial population is evaluated with assigned MOEA *driver* (e.g. NSGAII). After reaching specified progress ratio between 2 last metaepochs,

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

HGS may create additional sprouts – MOEA nodes with smaller populations created basing on the most promising individuals. On final stages of computing, MO-mHGS may consist of multiple sprouts connected in tree-like node structure. Each sprout can be treated as separate, independent unit. This specific calculation structure can be naturally transformed into actor-based model with parallel execution capabilities. Therefore, our goal was to adapt and optimize MO-mHGS to meet the actor model's requirements. It resulted in creating new improved model, Multiobjective Optimization Distributed HGS (denoted as MO-DHGS or DHGS).

The key problem was to identify types of actors and messages exchanged between them without loosing basic flow of MO-mHGS algorithm. Eventually, we defined two kinds of actors: Supervisor and Node. The actor system consist of one Supervisor and multiple Nodes that can be added dynamically during sprouting procedure. Supervisor should manage metaepochs and ensure parallel execution of Nodes is limited by the algorithm flow. Node is just an equivalent of single HGS sprout. At the beginning of simulation Supervisor creates single (root) node. Supervisor communicates with Nodes by passing messages that may contain additional data. There are several messages types used during single metaepoch:

- *GetState* – asking node about it current HGS state (alive/dead), maturity, etc;
- *StartMetaepoch* – triggering new epoch's calculation in each node that receive this message;
- *GetPopulation* – asking a node about current population. Some of this messages may be used on different stages of computing, but we focused on minimizing unnecessary data exchange due to its impact on the performance.

Apart from architectural changes, we also had to modify the control flow. Core MO-mHGS procedure was left intact, so a single DHGS step can be described as the same sequence of phases: 1) Run Metaepoch 2) Trim Not Progresssing Sprouts 3) Trim redundant sprouts 4) Release sprouts 5) Revive dead tree. However, each phase was adapted to the actor model. In case of phases 2-5 it required major changes, because these procedures require exchanging a lot of information between supervisor and nodes (or even between nodes). In classical approach, where whole memory was shared between all units, it could be easily achieved. In case of asynchronous model it often leads to messages overload or disturbing order of processed data. It may not only have negative impact on performance, but also violate the algorithm assumptions.

Algorithm 1 shows new, actor-based, supervised sprouting procedures. In classical approach, sprouting was simple, recursive procedure. Every node asked its old sprouts to perform sprouting and, at the end, released new sprouts if constraints (ξ – *sproutivness*, amount of children created after a metaepoch and ξ_{max} – maximum number of living children in a tree node) had been preserved. In actor-based approach there are two major problems that need to be addressed. First of all, in order to preserve sproutivness, node has to filter out alive sprouts. It can be achieved by sending *GetState* message to each sprout and wait for all

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

Algorithm 1 Supervisor Sprouting Task

```
1: procedure InitSprouting(hgsNodes)
2:     currentLevelStates ← ∅
3:     nextLevelStates ← ∅
4:     lvl ← FindLeafLevel(hgsNodes)
5:     i ← 0
6:     node ← hgsNodes_{lvl,i}
7:     Send(node, "SproutingRequest", nextLevelStates)
8: end procedure
9:
10: procedure ReceiveSproutingAnswer(node, state, sproutStates)
11:     currentLevelStates ← currentLevelStates ∪ {state}
12:     nextLevelStates ← currentLevelStates ∪ sproutStates
13:     i ← i + 1
14:     if i ≥ |hgsNodes_{lvl}| then
15:         lvl ← lvl − 1
16:         i ← 0
17:         nextLevelStates ← currentLevelStates
18:         currentLevelStates ← ∅
19:         if lvl < 0 then
20:             return                          ▷ End of sprouting procedure
21:         end if
22:     end if
23:     node ← hgsNodes_{lvl,i}
24:     Send(node, "SproutingRequest", nextLevelStates)
25: end procedure
```

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

responses at the beginning of procedure. There is guarantee that state can't be changed at this point of the algorithm so it suffices to do it once per sprouting procedure. The second issue is more difficult to handle. Comparing best individuals (Γ) to choose seed of new sprout's population requires gathering information from all next level nodes (not only sprouts of the current node). If we let each node to perform sprouting asynchronously, new sprouts will impact comparison procedure of other nodes. Depending of gathering time the results may vary – some nodes might have finished sprouting, others have not even started. This is why the whole sprouting procedure in actor-based version should be supervised. Our approach is to divide sprouting into 2 tasks. First task is conducted by the Supervisor. It runs sprouting sequentially traversing all nodes from a tree level, starting with leaves. Supervisor gathers information about sprout states that may be used in comparison procedure. After receiving sprouting summary from a node that finished its procedure, supervisor updates *nextLevelStates* set with states of newly created sprouts. This set is later passed to another node starting sprouting. It also keeps updating *currentLevelStates* that eventually becomes *nextLevelStates* when supervisor goes to a lower tree level. That's how we ensure every node operates on the most current knowledge about HGS tree. Moreover, each node sends this information to supervisor only once so that we can minimize required communication.

Sprouting procedure is the most complex (and important) part of HGS. The remaining HGS procedures also required adjusting to the new model, but usually solutions were similar to described idea and other differences lie mainly in implementation details. Full implementation of DHGS can be found in Evogil project, our open-source evolutionary computing platform [2].

4.2 Evaluation methodology

In order to measure hypothetical impact of parallel HGS nodes on overall MO-DHGS performance, we have measured the cost of calculations. Cost is expressed as number of fitness function evaluations. Each algorithm had the same cost constraints (*budget*). We assumed that in case of parallel execution a cost of single metaepoch can be reduced to cost of longest running node. In other words, for set of costs $\{c_1, c_2, ...c_n\}$ where c_i is a cost of metaepoch of a node i, the overall DHGS metaepoch cost can be expressed as $C_{DHGS} = max\{c_1, c_2, ...c_n\}$, while cost of sequential MO-mHGS is $C_{HGS} = \sum_{i=1}^{n} c_i$.

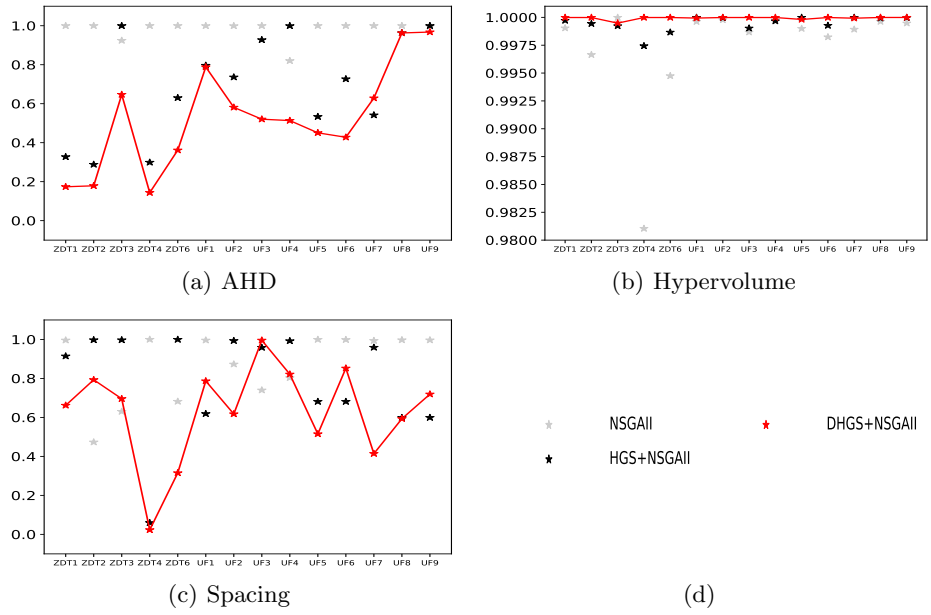
All runs of the system were performed on Evogil platform with the same algorithms' parameters as in our previous research [9]. This time we focused on comparing 2 specific algorithms that were combined with MO-mHGS and DHGS models: classical dominance-based approach NSGAII [4] and a hybrid of MOEA and particle swarm optimizer, OMOPSO [15]. In the later case, choice was dictated by promising results of previous research, where OMOPSO characteristics have proved to be well fit to HGS meta-model. Algorithms were evaluated on ZDT and CEC09 benchmark families.

[2] https://github.com/Soamid/evogil

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

Cost and error scaling on different tree levels were also adjusted as in our previous work, simulating real-world problems behaviour. Again, we chose to take into consideration only the best working set of parameters, thus cost modifiers were set to $\langle 0.1, 0.5, 1.0 \rangle$ and error variation levels to $\langle 10\%, 1\%, 0.1\% \rangle$. These parameters were configured for both MO-mHGS and MO-DHGS.

4.3 MO-DHGS results



(a) AHD (b) Hypervolume

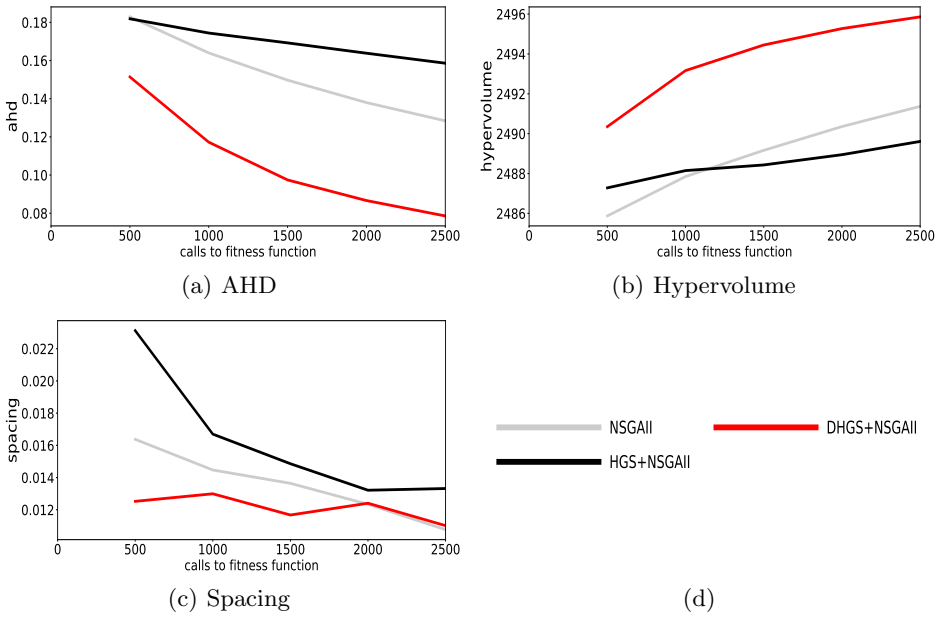
(c) Spacing (d)

Fig. 2. Run summary (budget=2500, NSGAII hybridization). Markers represent algorithms' normalized metric outcomes at final stage of system run. MO-DHGS outcomes are connected with lines.

For each result set we have evaluated several quality indicators. In this paper we focus on three popular metrics: Average Hausdorff Distance (AHD)] [14] (combination of GD and IGD metrics), Hypervolume [5] and Spacing [13]. Figure 2 shows summary of results from end of the system run for first considered algorithm, NSGAII. It is clear that DHGS improves performance of single-deme NSGAII, but also almost always wins with classical MO-mHGS regardless of metric. In terms of distance from Pareto front and individuals distribution there are significant differences between HGS methods, up to 50% of AHD value (UF4).

If we take a closer look on UF4 problem case and distribution of metric value over consumed budget (Fig 3), we can observe that DHGS achieves good results also on earlier stages of system run. However, DHGS converges much faster

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13



(a) AHD (b) Hypervolume

(c) Spacing (d)

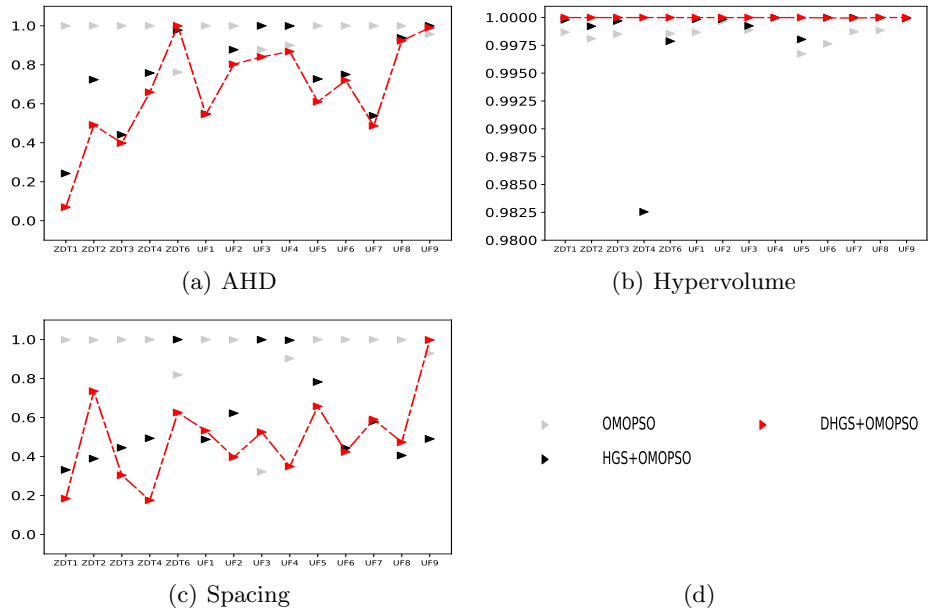
Fig. 3. Detailed results of NSGAII hybridization for all considered quality indicators tackling UF4 benchmark.)

than MO-mHGS and gap between these two increases with time. It is related to parallel architecture of DHGS: at the beginning it operates on single root node. Additional nodes (evaluated in parallel) are created later, after reaching satisfying progress ratio on lower level nodes.

Similar behaviour can be observed during OMOPSO computing. Summary of results across all benchmark problems (Fig 4 lead to the same conclusions: DHGS outperforms other solutions in vast majority of cases. Detailed charts of representative example (UF5, Fig 5 present how taking advantage of parallelism may eventually provide tremendous results. It is worth to note that in case of UF5 problem DHGS struggled to beat MO-mHGS in first half of the process. Later (after reaching promising outcomes in root node) it released sprouts, pushing final results farther and resulting in higher Hypervolume. It also found better individuals in terms of AHD after reaching about 60% of the total budget (even in comparison to final MO-mHGS results). This observation is important for our research, as MO-mHGS with OMOPSO driver was our best hybridization attempt in the previous research. This version of algorithm always reached optimal values very fast.

In order to summarize DHGS experiments, the outcomes we calculated score for each algorithm at two stages: after reaching budget equal to 1000 fitness evaluations and at the end of system run (2500 evaluations). Score was calculated basing on number of strong and weak wins for a given problem and budget step.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13



(a) AHD (b) Hypervolume

(c) Spacing (d)

Fig. 4. Run summary (budget=2500, OMOPSO hybridization). Markers represent algorithms' normalized metric outcomes at final stage of system run. MO-DHGS outcomes are connected with lines.

We define weak win as a situation when specified algorithm is no worse than all other solutions with some arbitrary chosen confidence value. In our experiments we set the confidence to 0.5% for Hypervolume and 5% for all other metrics. Algorithm is marked as strong winner if there are no weak winners for the considered result set. At the end, each strong winner obtains 2 points and weak winner scores 1 point. Table 1 contains summarized values of strong and weak wins in all considered situations. Again, DHGS versions of algorithms gather best scores in all cases. In multiple examples DHGS score value is improved at the end of system run (with one notable exception of NSGAII spacing: DHGS score decrases from 24 to 14 while single-deme NSGAII gahters some points).

5 Evaluating DHGS in asynchronous environment

Results presented in previous section should be considered as hypothetical. They are based on several assumptions about possibility of fitness adjustment and cost-free parallel execution. Real world problems results will be determined by proper model configuration, but also by technical conditions. In order to test DHGS model we implemented realistic multi-process version of the algorithm. We used thespian and rxPy Python libraries to include DHGS as new Evogil platform component. Then we created new simulation mode: to measure realistic

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

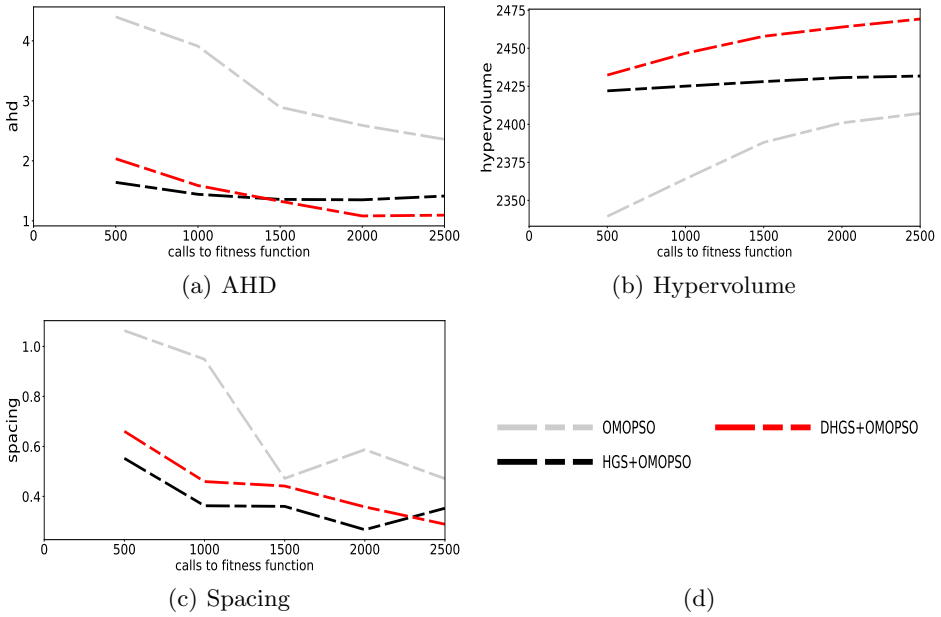


Fig. 5. Detailed results of OMOPSO hybridization for all considered quality indicators tackling UF5 benchmark.)

performance we applied time constraints instead of budget. During simulation we have measured time of each metaepoch and sampled current results every 2 seconds. Whole simulation had 20s timeout.

Note that all presented simulations were run without fitness evaluation adjusting – these are the same benchmark problems that were used in previous simulations, but they evaluate fitness the same way on every HGS node, regardless of tree level. That's why in time-bound tests even MO-mHGS ends up with worse results than single-deme algorithm.

All experiments were conducted on Windows 10 with Intel Core i9-9900K (3.6GHz with 8 cores and 16 threads) and 16GB RAM.

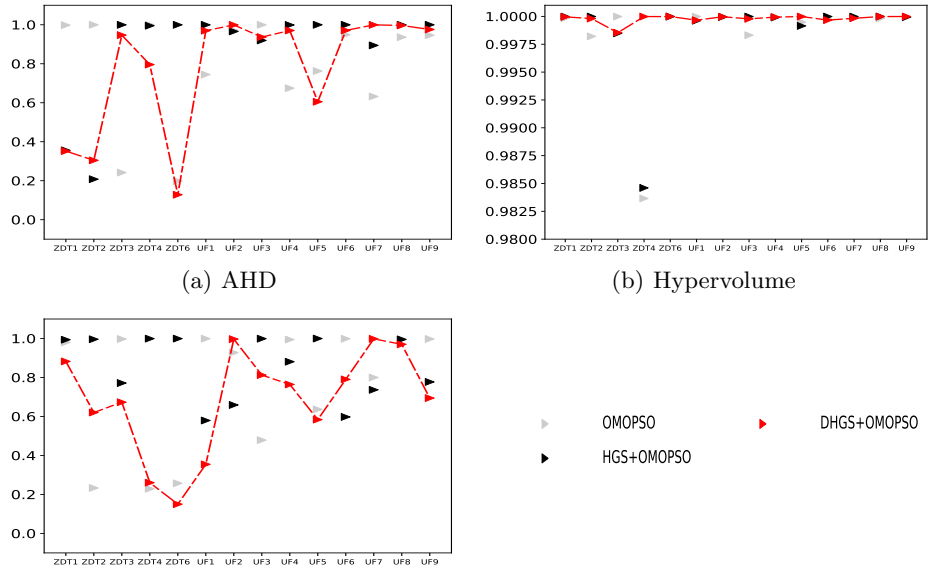
As predicted, results summary of realistic OMOPSO hybridization (Fig 6 look different than its simulated version. DHGS still outperforms MO-mHGS and single-deme algorithm in most situations, but it happens less frequently. Especially in terms of spacing indicator final results do not seem to have one dominating solution. On the other hand, basic AHD metric looks much better: usually asynchronous DHGS beats at least one of its competitors and almost always improves synchronous MO-mHGS results.

In fact, there are to separate sources of the problem here. First of all, the lack of aforementioned fitness evaluation adjusting that can be implemented in real-world problems, but it is not applicable to popular benchmark problems. That's why DHGS (and MO-mHGS) can't beat NSGAII performance even though both

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

Table 1. Scores of NSGAII and OMOPSO system in all benchmark problems. For each metric and budget stage, all wining methods are shown. Values in parentheses represent methods' scores. If method is not present in a cell, its score is 0.

	1000	2500
ahd	**DHGS+NSGAII(22)**, HGS+NSGAII(6)	**DHGS+NSGAII(23)**, HGS+NSGAII(5), NSGAII(2)
hypervolume	**DHGS+NSGAII(15)**, HGS+NSGAII(13), NSGAII(2)	**DHGS+NSGAII(18)**, HGS+NSGAII(9), NSGAII(5)
spacing	**DHGS+NSGAII(24)**, HGS+NSGAII(4)	**DHGS+NSGAII(14)**, HGS+NSGAII(7), NSGAII(7)
ahd	**DHGS+OMOPSO(15)**, HGS+OMOPSO(13), OMOPSO(1)	**DHGS+OMOPSO(22)**, OMOPSO(4), HGS+OMOPSO(2)
hypervolume	**DHGS+OMOPSO(17)**, HGS+OMOPSO(11), OMOPSO(2)	**DHGS+OMOPSO(19)**, HGS+OMOPSO(9), OMOPSO(3)
spacing	**DHGS+OMOPSO(14)**, HGS+OMOPSO(12), OMOPSO(2)	**DHGS+OMOPSO(17)**, HGS+OMOPSO(9), OMOPSO(2)



(a) AHD (b) Hypervolume

(c) Spacing (d)

Fig. 6. Realistic simulation summary (time=20s, OMOPSO hybridization). Markers represent algorithms' normalized metric outcomes at final stage of simulation. MO-DHGS outcomes are connected with lines.

were clearly better in simulated version of our experiment. Moreover, original

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

reviving procedure does not allow us to remove dead nodes (or reuse their processes for creating new nodes), as they can be bring back to life later. Therefore, if the algorithm releases many sprouts in specific problem case, there will be also many processes. Obviously, it has negative impact on performance of whole simulation. This interesting observation leaves space for further research of adjusting number of processing units or improving HGS reviving procedure.

6 Conclusion

Striving toward construction of efficient metaheuristics can help in reasonable utilization of nowadays very popular supercomputing facilities. Moreover, focusing on high-level programming languages makes possible for the user to focus on the algorithm itself, relying on the technology supporting the development process.

In this paper we have shown, that actor-based DHGS model can be a natural improvement of MO-mHGS. Tree-like structure of HGS concept gave us opportunity to create generic, multi-deme asynchronous algorithm able to outperform its synchronous version and converge faster on later stages of the system run.

We have shown that the proposed solution is sensitive to environment conditions and implementation details nodes which leaves room for further investigation.

Nevertheless, in combination with fitness evaluation adjusting mechanism, DHGS is a powerful tool for solving real-world multi-objective problems.

In the future we will focus on experimenting scalability in broader range, which seems to be promising because of the actual actor model used for synchronization of the systems work.

References

1. Bäck, T., Schwefel, H.P.: Evolutionary computation: An overview. In: Fukuda, T., Furuhashi, T. (eds.) Proceedings of the Third IEEE Conference on Evolutionary Computation. IEEE Press (1996)
2. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (2000)
3. Ciepiela, E., Kocot, J., Siwik, L., Dreżewski, R.: Hierarchical approach to evolutionary multi-objective optimization. In: Computational Science–ICCS 2008, pp. 740–749. Springer (2008)
4. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Lecture notes in computer science **1917**, 849–858 (2000)
5. Fonseca, C.M., Paquete, L., López-Ibánez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. pp. 1157–1163. IEEE (2006)
6. Gajda-Zagorska, E., Smolka, M., Schaefer, R., Pardo, D., Álvarez Aramberri, J.: Multi-objective hierarchic memetic solver for inverse parametric problems. Procedia Computer Science **51**, 974–983 (2015)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13

7. Gajda-Zagórska, E.: Multiobjective evolutionary strategy for finding neighbourhoods of pareto-optimal solutions. In: Esparcia-Alcázar, A. (ed.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 7835, pp. 112–121. Springer Berlin Heidelberg (2013)
8. Krzywicki, D., Turek, W., Byrski, A., Kisiel-Dorohinicki, M.: Massively concurrent agent-based evolutionary computing. J. Comput. Science **11**, 153–162 (2015). https://doi.org/10.1016/j.jocs.2015.07.003, https://doi.org/10.1016/j.jocs.2015.07.003
9. Lazarz, R., Idzik, M., Gadek, K., Gajda-Zagorska, E.: Hierarchic genetic strategy with maturing as a generic tool for multiobjective optimization. Journal of Computational Science **17**, 249–260 (2016)
10. Martin, W.N., Lienig, J., Cohoon, J.P.: Island (migration) models: evolutionary algorithms based on punctuated equilibria. Handbook of evolutionary computation **6**(3) (1997)
11. Schaefer, R., Adamska, K., Telega, H.: Clustered genetic search in continuous landscape exploration. Engineering Applications of Artificial Intelligence **17**(4), 407–416 (2004)
12. Schaefer, R., Kolodziej, J.: Genetic search reinforced by the population hierarchy. In: Foundations of Genetic Algorithms. vol. 7, pp. 383–401 (2002)
13. Schott, J.R.: Fault tolerant design using single and multicriteria genetic algorithm optimization. Tech. rep., DTIC Document (1995)
14. Schütze, O., Esquivel, X., Lara, A., Coello, C.A.C.: Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. Evolutionary Computation, IEEE Transactions on **16**(4), 504–522 (2012)
15. Sierra, M., Coello Coello, C.: Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance. In: Coello Coello, C., Hernández Aguirre, A., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, vol. 3410, pp. 505–519. Springer Berlin Heidelberg (2005)
16. Starzec, M., Starzec, G., Byrski, A., Turek, W.: Distributed ant colony optimization based on actor model. Parallel Computing **90** (2019). https://doi.org/10.1016/j.parco.2019.102573, https://doi.org/10.1016/j.parco.2019.102573
17. Stypka, J., Turek, W., Byrski, A., Kisiel-Dorohinicki, M., Barwell, A.D., Brown, C., Hammond, K., Janjic, V.: The missing link! A new skeleton for evolutionary multi-agent systems in erlang. International Journal of Parallel Programming **46**(1), 4–22 (2018). https://doi.org/10.1007/s10766-017-0503-4, https://doi.org/10.1007/s10766-017-0503-4
18. Turek, W., Stypka, J., Krzywicki, D., Anielski, P., Pietak, K., Byrski, A., Kisiel-Dorohinicki, M.: Highly scalable erlang framework for agent-based metaheuristic computing. J. Comput. Science **17**, 234–248 (2016). https://doi.org/10.1016/j.jocs.2016.03.003, https://doi.org/10.1016/j.jocs.2016.03.003

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50420-5_13