Distributed Resource Allocation as Co-Evolution Problem

Sven Tomforde, David Meier, Anthony Stein, and Sebastian von Mammen
Organic Computing Group, Universität Augsburg
Eichleitnerstr. 30, 86159 Augsburg, Germany
[sven.tomforde | david.meier | anthony.stein | sebastian.von.mammen]@informatik.uni-augsburg.de

***Abstract*—Distributed self-organising systems often face conflicts if more than one entity tries to access a limited resource. In order to solve this conflict, research focuses on techniques for resource allocation considering different priorities. In this paper, we propose to tackle the decision problem of whom to assign the resource by means of a co-evolutionary approach. We investigate appropriate fitness estimations, representation schemes, and configuration of the underlying genetic operators. We demonstrate the convergence and efficiency of our approach using an exemplary system model.**

I. Introduction

It is a common problem in distributed, self-organising systems that a certain resource might be required by several entities at the same time. This problem becomes all the more challenging with an increasing degree of self-organisation, i.e. in systems with a maximally distributed control mechanism [1]). In order to address this problem, decisions are typically made to give priority to individual entities based on (1) distinct features such as unique identifiers, (2) random decisions, or (3) scheduling principles such as First-Come-First-Serve, cf. e.g. [2]. Tournaments pose an alternative for such a decision mechanism. Here, two or more entities compete in certain tasks and the winner receives the right to access the resource. In this paper, we develop and investigate a co-evolutionary approach to such tournament-based conflict resolution scheme for resource allocation in self-organising systems.

Smart camera networks [4] represent a common research domain considered by the field of Organic Computing [3], where the outlined resource conflict in distributed, self-organising systems arises. A smart camera (SC) is a limited resource in terms of tasks to be accomplished: At any given point in time, it might either (i) observe certain areas to detect intruders, (ii) track a certain person's movements, or (iii) participate in a 3D-reconstruction of an object with other SCs. All of these tasks compete for the same SC, whereas a conflict resolution policy is usually not in place (be it assigned priorities, computable criteria or simple decision logic).

We propose to model the decision problem as a game played by two opponents competing for the same resource. Their odds are identical to begin with, but one of the opponents' strategies will come out on top. In order to find optimal strategies, we rely on co-evolution.

The remainder of this paper is organised as follows: Section II gives a brief overview of the state-of-the-art. Section III introduces our model and the design of the underlying co-evolution scheme. We present an evaluation of our approach in Section IV. This includes an analysis of the most successful strategies. Finally, Section V summarises the paper and gives an outlook to future work.

II. Related Work

A. Co-Evolution

Co-evolution is a concept known from biology where "the change of a biological object is triggered by the change of a related object" [5]. Conceptually, this means that successful modifications in the genome of a certain species affect the evolution of the genome of another species and vice versa. This observation from nature has been transferred to technical systems, initially to explain the development in software and hardware [6]. For instance, changes in hardware have been shown to bring about novel developments and functionalities in software applications, and novel possibilities in software allow for more flexible hardware architectures. A similar interpretation has been discussed in the context of socio-technical systems, where a "joint optimisation" between systems analysis and system design has been considered [7].

From an algorithmic point of view, co-evolution can be considered an aspect of optimisation strategies, and a field within the wider area of evolutionary computation. In general, two concepts are distinguished: cooperative and competitive co-evolution. Cooperative co-evolution tackles an optimisation problem (typically a large and complex problem) by dividing it into "subcomponents" (also called "species") to be solved individually [8]. Thereby, each of these subcomponents is mapped on an individual population. The evaluation is done cooperatively for each individual of the populations. The concept imitates the observation from nature that individuals of a particular group of species mate [9].

In contrast, competitive co-evolution tries to imitate an "arms race". Here, the individuals' fitnesses are determined in tournaments rather than by means of an absolute fitness measure. As a result, an increased fitness value of one candidate solution implies a decreased fitness value of another. In a perfect setting, this results in an urge to continuously find better solutions than the opponent [10], [11]. Competitive co-evolution has been successfully applied to learning [12], sorting networks [13], or finding vulnerabilities in an organisation [14]. In this paper, we focus on such a competitive co-evolution process.

B. Resource Allocation

Traditional approaches to the research allocation problem come from the scheduling domain, see e.g. [15]. For solving the problem, they consider either the arrival time (such as first-in or last-in is served), static priorities, or dynamic priorities (e.g. in terms of deadlines).

Besides such pre-defined decision schemes, resource allocation is often subject to optimisation, which can be broadly seen as a combinatorial problem. Accordingly, the idea is to identify an operation schedule that is consistent with resource constraints and to find the best possible trade-off given a certain utility function. In literature, several according concepts can be found, going back to the 1960s, see e.g. [16]. Several approaches have been discussed that aim at finding optimal solutions. Examples are dynamic programming or branch-and-bound concepts [17]. Due to complexity and scalability issues they are rarely applied to real-world problems [18].

As an alternative to optimal solutions, good solutions are often acceptable, which allows the use of heuristics to arrive at satisfactory solutions without exhaustively sampling the search space. Heuristic rules are easy to understand, to design, and computationally efficient to apply [19]. In the context of project management, heuristic rules were designed to achieve general goals such as “least total slack” or “earliest late start” [20]. But the rules would not adapt to specifics of the particular problem instance and hence result in sub-optimal behaviour.

Optimisation heuristics, however, are able to identify satisfactory problem-specific solutions. There is a variety of optimisation heuristics that can be applied to the resource allocation problem, ranging from standard hill climbing to evolutionary techniques and swarm-based methods, cf. e.g. [21]. All these concepts typically rely on a centralised algorithmic component and are therefore not efficiently applicable to the distributed allocation problem we are facing in the context of self-organisation and Organic Computing.

III. Approach

A. System Model

We model the decision process for the resource allocation problem by means of a simple game. In this game, two opponents play an odd number of rounds – the player with more successful rounds wins the game. Initially, each player is given a stick of a certain length, whereas the length represents the player’s amount of resources. In each round, both players have to cut off a piece of their resource, without knowing which length the opponent is ready to sacrifice. The player who cut off the greater piece wins the round (Figure 1).

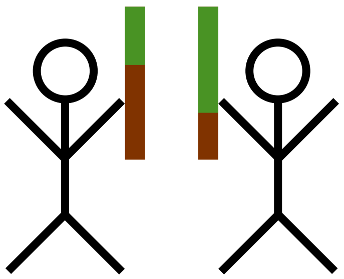


Fig. 1. The figure illustrates the setup of the game: Two players decide about their resource utilisation (green part). Since the right player cuts off the larger part, it wins this round.

Obviously, the setup of the game does not favour one opponent over the other. It is also not influenced by the choice of who begins. There are only two possibilities for a draw: (a) Both players act constantly identically, and (b) both players won the same number of rounds and have the same fraction of the resource left for the last round. The probability for these cases can be influenced by the initial length of the stick and the minimal cut size (which, in turn, impacts the period of time a game can take).

We further discern that the resource-stick game has some advantages over other decision policies: (1) It does not favour one opponent, (2) it is fast to execute, (3) each opponent can optimise his strategy by playing against himself, and (4) it comes to a decision with a very high probability.

B. Approach based on Co-Evolution

We solve the resource-stick game by means of co-evolution. To this end, we consider two players, A and B, that both have a set (population) of strategies (individuals or candidate solutions) available. θ denotes the number of individuals of each population. Both players repeatedly compete with each other playing the resource-stick game. The players’ strategies co-evolve stepping through the following algorithm.

1) **Step 1:** Populations A and B are initialised with θ randomly generated candidate solutions.
2) **Step 2:** Player B randomly selects one of his strategies, his currently best candidate.
3) **Step 3:** Player A lets all his strategies compete against B’s currently best candidate. The individual from A with the greatest fitness (measured by a function ϕ, considering e.g. the amount of resources left), becomes A’s currently best candidate.
4) **Step 4:** Player B lets all his strategies compete against A’s currently best candidate. The individual from B with the greatest fitness, e.g. the amount of resources left, becomes B’s currently best candidate.
5) **Step 5:** The worst ϵ individuals from both populations are deleted.
6) **Step 6:** To each population, ϵ new individuals are added, generated using genetic operations.
7) **Step 7:** Steps 3 to 6 are repeated until a pre-defined stopping criterion is fulfilled.

The basic algorithm contains decisions about the representation scheme of the individuals, the fitness function, and the genetic operations. We introduce the corresponding definitions next.

C. Representation Scheme

The representation scheme describes the genomic encoding of each individual. Technically, the genome can be understood as a vector of attributes defining the behaviour of an individual. In particular, a vector may comprise an entry for each round of the resource-stick game – encoding the fraction of the resource to be cut off the stick. As the representation scheme has great impact on the performance of the optimisation process, we consider two different ones (evaluation in Section IV).

1) Representation A (Integer):: The first representation scheme makes use of integer values, with each value coding an absolute percentage of the initial resource size to be spent in the particular round. Consider the following array of values as an example:

10	20	25	66	42	88	2

In this example, 10% of the initial resource is spent in the first round, 20% in the second round, and so on. As already incorporated in this example, the concept is prone to early resource loss (in case of sub-optimal decisions).

2) Representation B (Relative Coding):: As alternative to the absolute coding of resource utilisation, a relative scheme can be applied. Here, each entry in the vector defines which fraction (given in percent) of the available resource is utilised in the particular round. This decreases the possibility of spending the full resource, but has the drawback that it seldom makes use of the complete resource (i.e. a small fraction is left over, if the last entry does not equal 100%). Considering the strategy presented above, the genes would encode: Spend 10% of the resource for round 1, 18% for round 2 (i.e. 20% of remaining 90%), and so on.

D. Selection Strategy

The selection strategy decides which individuals are promoted to the next generation. In literature, a variety of approaches exist (see e.g. [22]). We decided to rely on an *Elitist* approach – meaning that a subset of the best individuals are promoted to the next generation in an unchanged manner. In other words, we explicitly collect and keep the best solutions over the course of all generations, which proves especially effective for evolving solutions to problems with static fitness functions.

However, we compare two different selection strategies, one of which considers the diversity of the chosen specimen: (1) A "standard" elitist selection algorithm choses the best s_{rate} $(1-\epsilon)$ individuals (see Algorithm 1). (2) A modified variant that chooses the best s_{rate} distinct individuals (see Algorithm 1). In both algorithms, the population size is given by the parameter p_{size}, while s_{rate} controls the fraction of elitists that are promoted to the next generation.

E. Genetic Operators

Besides choosing individuals for the next population, we need to generate new individuals for each generation to fill the population after deletion of the ϵ worst candidates. This is done using the genetic operators: selection, mutation, and crossover. We explain the configuration and interplay of these operators in the following paragraphs.

Algorithm 1 Selection Algorithm

```
1: procedure SELECT($P^A$)
2:     sort population $P^A$ by decreasing fitness
3:     $P^A \leftarrow$ choose the first $\lfloor p_{size} * s_{rate} \rfloor$ elements from $P^A$
4:     return $P^A$
5: end procedure
```

Algorithm 2 Alternative selection algorithm

```
1: procedure SUPERSELECT($P^A$)
2:     sort population $P^A$ by decreasing fitness
3:     $P^A \leftarrow$ choose the first $\lfloor p_{size} * s_{rate} \rfloor$ elements from $P^A$ that are distinct from those that are already chosen
4:     return $P^A$
5: end procedure
```

1) Parent Selection: When generating new individuals, we first have to select *parent individuals* for procreation. Again, a variety of approaches to this selection procedure are known from literature. We decided to use a roulette-wheel approach that considers each remaining individual in relation to its fitness, since this seems to be the most commonly accepted variant [22].

2) Crossover: The selected parents provide the genetic basis for the offspring. Individual genes from either parent are combined to form a new individual by means of the so-called crossover, or recombination, operator, which imitates natural sexual reproduction. Implementations range from one-point crossover to multi-point crossover. The former splits and recombines the genetic encoding at one randomly chosen point, the latter recombines several segments whose positions and lengths often follow a uniform probability distribution. For our experiments, we implemented one-point, two-point and uniform crossover operators [23].

3) Mutation: The mutation operator introduces small modifications into the offspring's genome. As such, it adds an exploration component to the search for the best candidate solution [23]. In Equation 1, we define the mutation function $M(x)$. Within this equation, $x \in [x_{min}, x_{max}]$ is the initial manifestation of a gene and $M(x)$ is the corresponding value after mutation has been applied. The range of x is delimited by upper (x_{max}) and lower (x_{min}) boundaries – which reduces the decision space. M_{max} defines a maximally allowed deviation of the mutated allele in comparison to the initial allele. M_{factor} describes the fraction of the mutated parts in the final allele. The random variable $r_{pos} \in [0, 1]$ determines, whether a mutation is applied or not. The specific deviation of the allele is estimated using the random variable $r_{int} \in [0, M_{max}]$. This random variable is uniformly distributed, resulting in lower probabilities for extrema but higher probabilities for average values.

$$M(x) = \begin{cases} x_{max} & \text{if } xr_{int} \geq x_{max} \land r_{pos} \leq M_{factor} \\ x_{min} & \text{if } xr_{int} \leq x_{min} \land r_{pos} \leq M_{factor} \\ xr_{int} & \text{if } xr_{int} \in [x_{min}, x_{max}] \land r_{pos} \leq M_{factor} \\ x & \text{otherwise.} \end{cases} \tag{1}$$

F. Fitness Estimation

The fitness function allows us to quantify the performance of an individual/solution. Following the system model as introduced in Section III-A, the overall objective is to win the game. This basic criterion can be augmented considering the following two approaches.

1) High Resource Utilisation: The first approach combines the concept of winning a round with a desired degree of resource utilisation, i.e. the sub-goal is to maximally deplete a given resource (Eqn. 2).

$$F(r_{win}, s_{rem}) = \begin{cases} 1 + \lambda \frac{r_{win}}{r_{all}} + (1-\lambda)\left(1 - \frac{s_{rem}}{s_{init}}\right) & \text{if win,} \\ \lambda \frac{r_{win}}{r_{all}} + (1-\lambda)\left(1 - \frac{s_{rem}}{s_{init}}\right) & \text{other.} \end{cases} \tag{2}$$

In this formula, $\lambda \in [0, 1]$ is the weighting factor favouring the number of successful rounds over resource utilisation (the *rounds weight factor*). Mathematically, the goal is to decrease the remaining resource to zero. r_{all} represents the number of rounds played and r_{win} the number of rounds won. The variable s_{rem} denotes the remaining resource after the last round and s_{init} defines the initial resource length. Consequently, the term $1 - \frac{s_{rem}}{s_{init}}$ increases with the utilisation of resources and is, thus, used to determine the resulting fitness value.

2) Low Resource Utilisation: The goal of the second approach to fitness calculation is contrary to the previous one. While trying to win as many rounds as possible, the player avoids too high resource utilisation (Eqn. 3). The variables are named in accordance with Eqn. 2.

$$G(r_{win}, s_{rem}) = \begin{cases} 1 + \lambda \frac{r_{win}}{r_{all}} + (1-\lambda)\frac{s_{rem}}{s_{init}} & \text{if win,} \\ \lambda \frac{r_{win}}{r_{all}} + (1-\lambda)\frac{s_{rem}}{s_{init}} & \text{other.} \end{cases} \tag{3}$$

3) Application of Fitness: For both augmented fitness functions, $F, G \in [0, 2] \forall r_{win}, s_{rem}$ holds. In order to calculate the overall fitness of a player, we multiply the fitness values of the played games as shown in Equation 4.

$$f_j = \frac{\gamma f_{j-1} + H(r_{win_j}, s_{rem_j})}{\gamma + 1} \tag{4}$$

The fitness in game j is recursively defined. The initial value is set to $f_0 = 0.5$. H denotes the method for calculating the fitness, and r_{win_i} represents the number of rounds won in game i. Analogously, s_{rem_i} is the remaining resource in game i. γ is the *fitness appliance factor*, which controls how the fitness of the current game is weighted in relation to the the current fitness of the player.

IV. Evaluation

In this section, we evaluate our concept and investigate the most appropriate configurations for the introduced control variables.

A. Experimental Setup

All simulations are performed 20 times, the figures show averaged values and standard deviation. Table I lists the considered standard parameter settings. Some of these settings are altered throughout the experiments, which we will explicitly state whenever applicable. In addition to the parameters introduced in Section III, the parameters of a single game have to be specified. This includes (1) the *number of rounds* the game is played (one player needs more than half of these rounds to win the overall game), (2) a fraction, the so-called *Minimum Decision*, of the resource that has to be utilised at each round (this refers either to a percentile or an absolute value depending on the method from Section III-C), and (3) an upper boundary of resources, or *Maximum Decision*, lost at each round.

TABLE I
Standard configuration of parameter settings throughout all experiments, if not indicated otherwise.

Variable	Configuration
Population size	20
Number of rounds	7
Resource length at start	1000
Generations	10000
Evolution type	Co-Evolution
Representation Scheme	Relative Coding
Minimum Decisions	1
Maximum Decisions	100
Selection rate	80 %
Selection method	Superselect
Crossover method	Probability Crossover
Crossover probability	60 %
Maximum Mutation	100 %
Mutation probability	5 %
Fitness calculation	High resource utilisation
Weight per round	0,9
Fitness factor	10

B. Experimental 1: Static Opponent

First, we demonstrate that the optimisation process works correctly. To this end, we modify the prescribed play by Player B as outlined in Section III-B. Instead of a responsive, adaptive behaviour, we now use a static strategy, i.e. player B always cuts off the same fraction of the resource at each round. In contrast, Player A optimises his population of solutions following the previously outlined evolutionary optimisation loop. Figure 2 illustrates the results. The figure emphasises the clear loss of the static player, losing about 92% of the rounds.

A slightly more challenging opponent changes his behaviour after losing a round. We implemented such a solution that chooses a randomly generated alternative strategy after losing and compared it against the evolutionary player. Figure 3 illustrates the results. We can observe that this strategy results

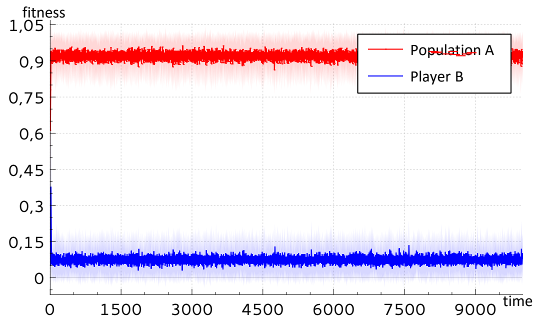


Fig. 2. Evaluation: Comparison of co-evolutionary player (A, red line) with a manually pre-optimised, static player (B, blue line).

in a constantly higher success rate (i.e. about 24%) but is still significantly outperformed by the evolutionary approach.

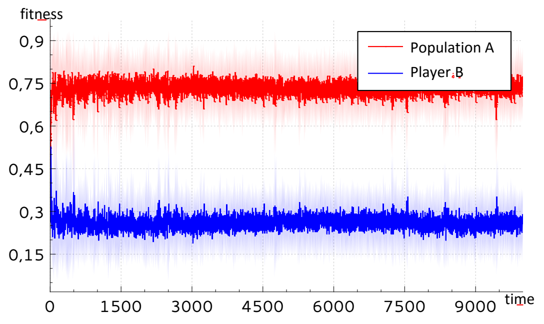


Fig. 3. Evaluation: Comparison of co-evolutionary player (A, red line) with a player that chooses a randomly generated alternative behaviour, if unsuccessful in the preceding round (B, blue line).

As a conclusion for Experiment 1, we can state the evolutionary approach works as intended and outperforms simple static solutions.

C. Experimental 2: Co-Evolution

In the second experiment, we compare the success of two populations that both follow the outlined evolutionary optimisation loop, i.e., we investigate the co-evolution scenario. Both opponents have the same chance to win and follow the same optimisation scheme. The difference lies in the choice of random variables. Consequently, we assume that the averaged behaviour converges towards similar results. Furthermore, we expect the standard deviation to decrease with higher generation numbers.

Figure 4 illustrates the achieved results. On average, both players win 50% of the games. We can observe that the standard deviation slightly decreases over generations. We can further observe that the co-evolutionary optimisation process leads to a periodic behaviour (expressed in the evolution of the standard deviation values): Although the average is almost constant at 50%, the standard deviation has a cyclic increase/decrease behaviour. This is due to the fact that as soon as one player finds a better solution, he is at an advantage for a certain number of rounds. The oscillation emerges from the

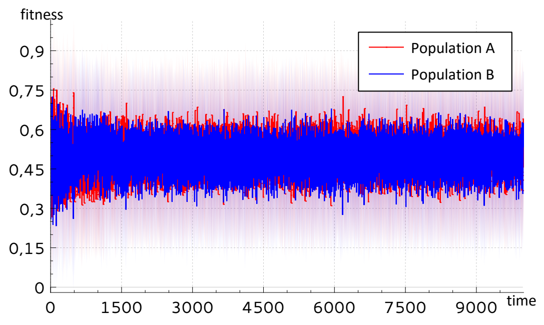


Fig. 4. Evaluation: Comparison of two co-evolutionary players.

need for adaptation by the losing player, and it illustrates the "arms race" mentioned in Section II-A.

Figure 5 illustrates the behaviour of both populations at the beginning of an optimisation run (these are values from one exemplary run, no averages). In Figure 5(a), the success rates of the populations are evaluated. We can see that this oscillates – but result in about 50% wins per player as expected. Figure 5(b) shows the success rate of the representatives (i.e. those individuals that are tested against the whole opposing population). Here, we can see a faster convergence compared to the entire population – this means that the process quickly identifies reasonably good candidates in both populations.

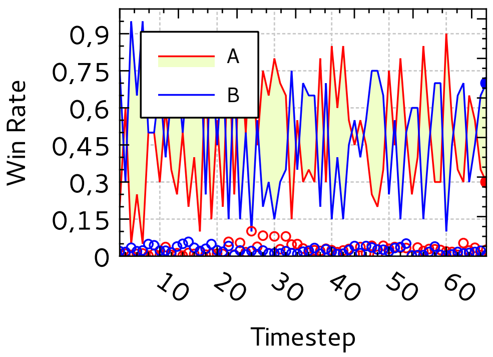
Figure 5(c) to 5(f) illustrate the decisions that are taken. The decisions are coded using a colour scheme: High resource utilisation (i.e. up to the maximum utilisation of 100%) are coded in red, while low resource utilisation (i.e. down to 1%) are coded in blue. The x-axis show the time for the last 65 generations, with 0 identifying the current game. The y-axis specifies the particular decisions to be taken (more precisely: the certain round of the game and its decision). One can see that the algorithms tends to start with medium resource utilisation decisions at the beginning (i.e. decision round 0) and decreases afterwards to lower utilisation decisions.

D. Experimental 3: Fitness Function

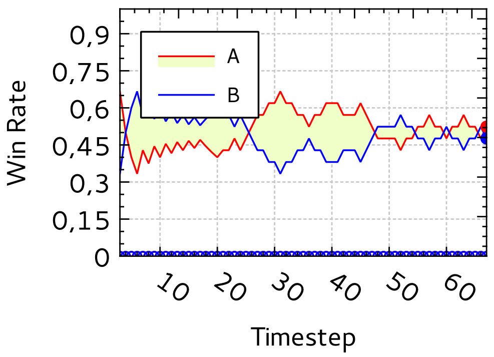
The next experiment analysis the behaviour considering the different fitness functions as introduced in Section III-F. Figure 6 illustrates the results using the absolute encoding scheme, while Figure 4 is based on relative encoding. Comparing both figures, we can observe that the relative encoding scheme has advantages, since it results in better converged solutions. Especially considering the standard deviation measured for both approaches, we can see that the absolute encoding scheme results in significantly higher values – which is an indicator for non-successful solutions.

E. Experimental 4: Mutation Probability

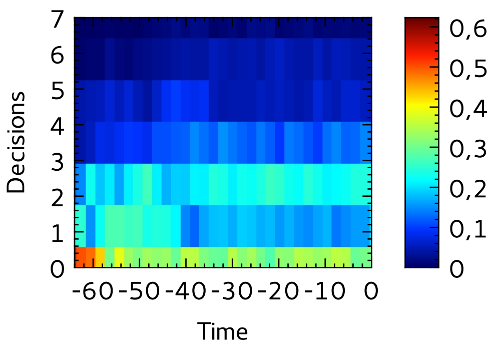
In Section III-E3, we defined two configuration parameters for the behaviour of the mutation strategy: (a) defining the maximum allowed mutation deviation and (b) defining the mutation probability. Figure 7 shows the results for the first parameter. We investigated a step-wise configuration of the



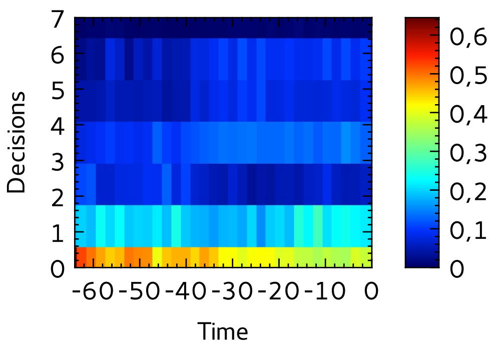
(a) Success rates for populations.



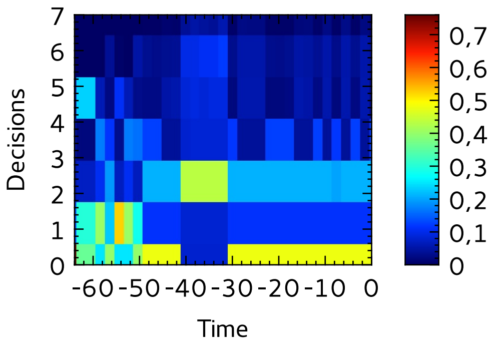
(b) Success rate for representatives.



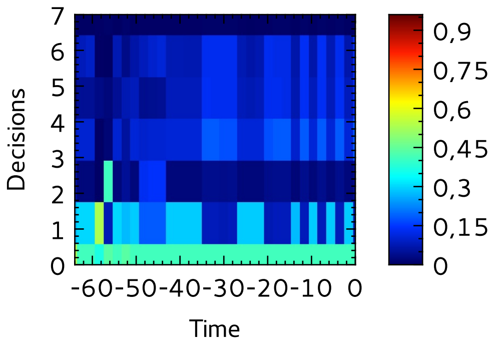
(c) Decisions taken by Population A.



(d) Decisions taken by Population B.



(e) Decisions taken by representative A.



(f) Decisions taken by representative B.

Fig. 5. Exemplary analysis of the first 70 rounds comparing the success rates (a and b) of representatives and populations. Part c to f illustrates the decisions taken by representatives and population: high resource utilisation is depicted in red, low resource utilisation in blue (see legend at the right side).

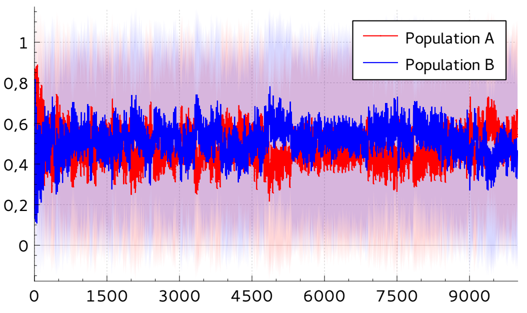
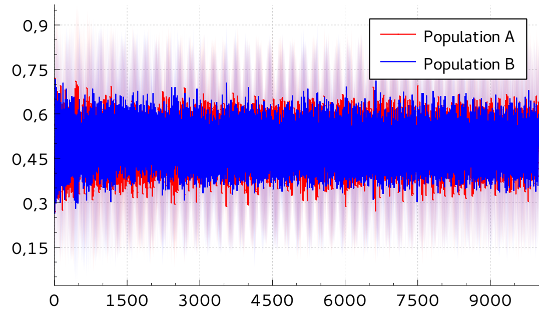


Fig. 6. Evaluation considering the absolute encoding scheme

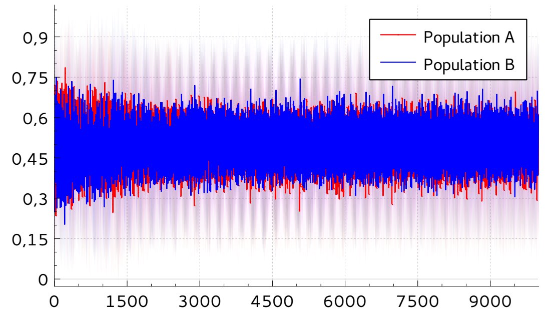
maximum mutation rate using 5% steps. The figure exemplarily shows the results for 20% and 40%. Again, the standard parameters as shown in Table I have been applied.

As illustrated by the figure, a higher maximum deviation due to mutation entails more generations towards convergent behaviour. In general, the parameter has only minor influences – within the shown range the behaviour is quite similar. Only to both extrema, the behaviour is influenced in a negative manner.

Figure 8 illustrates the results for varying the mutation rate. Again we analysed the behaviour based on steps of 5% intervals. Figure 8(a) investigates an exemplary mutation rate of 20% and Figure 8(b) of 50%. The difference is clearly visible: The higher the mutation rate, the lower the convergence (i.e. the higher the standard deviation). Again, we investigated a step-size of 5% and a configuration according to other work from the literature (see e.g. [14]) of 20% results in the best performance.

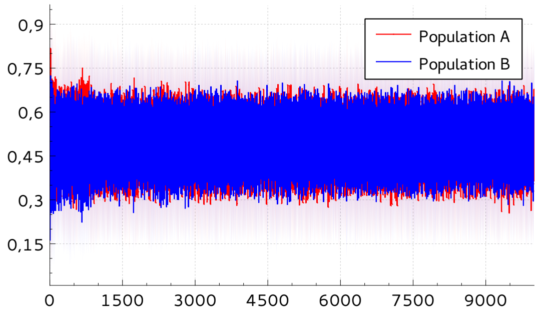


(a) Maximum Mutation: 20 %.

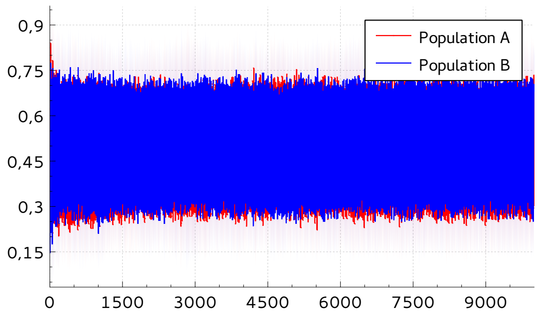


(b) Maximum Mutation: 40 %.

Fig. 7. Evaluation: Varying the maximum deviation rate due to mutation.



(a) Mutation Probability: 20 %.



(b) Mutation Probability: 50 %.

Fig. 8. Evaluation: Varying the mutation rate.

F. Experimental 5: Crossover

In Figure 4, we already used probability crossover. In contrast, Figure 9 shows the results when using two-point crossover. When comparing both results, we can see that the impact of the decision has only low impact: Both perform similar. The same holds for one-point crossover (not shown here).

G. Experimental 6: Population Size

As a last experiment, we analysed the impact of changing the population size. In general, a larger population size allows to transfer more "knowledge" to the succeeding generation. Figure 10(a) show the results for a population size of 10, while Figure 10(b) uses a very large size of 1000. As expected, the standard deviation decreases and the system benefits from more available diversity and knowledge with increasing population size. Hence, the decision about the size is mostly limited by storage, timing and computation boundaries.

V. Conclusion

This paper investigated a solution for a fair and efficient resource allocation scheme in distributed self-organised systems. The idea is to provide a playground in which jobs can "fight" for the resource. We modelled this as a two-player game and presented a co-evolutionary approach to i) self-optimise the own strategy by playing against oneself, and b) compete against each other for the resource and improve the behaviour over time as a result of competition.

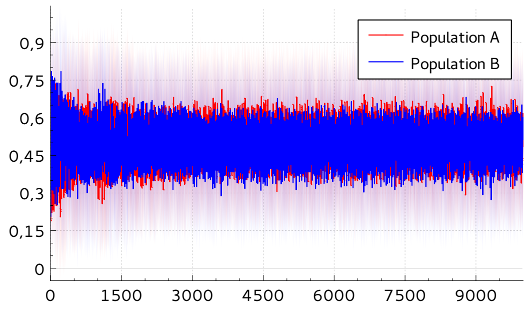
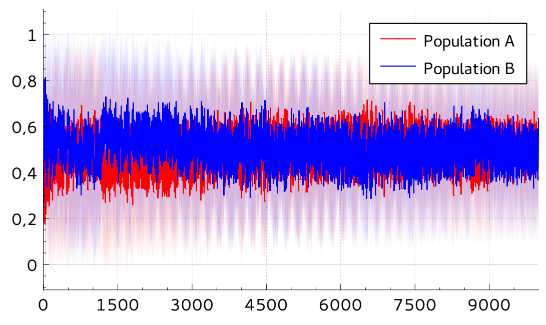


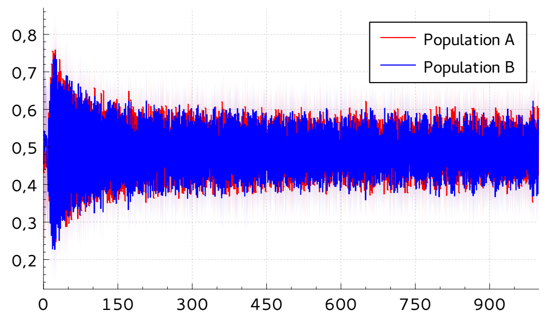
Fig. 9. Evaluation: Two-Point Cross-Over

The experimental evaluation compared different configurations of the possible strategies and control variables. Initially, we demonstrated that the approach works in the desired way by comparing the evolutionary strategy against two static concepts. Afterwards, we derived beneficial configuration settings for the considered variables.

Current work focuses on two aspects: Initially, we apply the approach to the resource allocation problem as introduced in the motivating example: smart camera control. We aim at demonstrating the fairness and efficiency of the developed solution in a real-world environment. In addition, we want to extend the concept towards more opponents. More precisely, the current approach covers only a decision problem with two opponents and we want to come up with a solution that is equally fast and reliable.



(a) Population size 10, 10000 generations



(b) Population size 1000, 1000 generations

Fig. 10. Evaluation of the impact of varying the population size. Different scale at y-axis due to different standard deviations.

References

[1] G. Muehl, M. Werner, M. Jaeger, K. Herrmann, and H. Parzyjegla, "On the Definitions of Self-Managing and Self-Organizing Systems," in *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference, Kommunikation in Verteilten Systemen, 26. Februar - 2. März 2007 in Bern, Schweiz*, Feb 2007, pp. 1–11.

[2] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*, 2nd ed. Prentice Hall, 2013, iSBN-13: 978-1292025520.

[3] C. Müller-Schloer, "Organic Computing: On the feasibility of controlled emergence," in *Second IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS Merged Conference), Sept. 8-10, 2004, Stockholm, Sweden*. ACM Press., 2004, pp. 2–5.

[4] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf, "The evolution from single to pervasive smart cameras," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, Sept. 2008, pp. 1–10.

[5] K. Yip, P. Patel, P. Kim, D. Engelman, D. McDermott, and M. Gerstein, "An integrated system for studying residue coevolution in proteins," *Bioinformatics*, vol. 24, no. 2, pp. 290 – 292, 2008.

[6] T. D'Hondt, K. D. Volder, K. Mens, and R. Wuyts, "Co-Evolution of Object-Oriented Software Design and Implementation," *TheKluwer International Series in Engineering and Computer Science*, vol. 648, no. 2, p. 207224, 2002.

[7] A. Cherns, "The principles of sociotechnical design," *Human Relations*, vol. 29, no. 8, pp. 783 – 792, 1976.

[8] M. A. Potter and K. De Jong, "A Cooperative Coevolutionary Approach to Function Optimization," in *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, ser. PPSN III. London, UK: Springer-Verlag, 1994, pp. 249–257.

[9] K. O. Stanley and R. Miikkulainen, "Competitive Coevolution Through Evolutionary Complexification," *J. Artif. Int. Res.*, vol. 21, no. 1, pp. 63–100, Feb. 2004.

[10] R. Dawkins and J. R. Krebs, "Arms races between and within species," *Proceedings of the Royal Society of London Series B,*, vol. 205, p. 489511, 1979.

[11] C. D.Rosin, "Coevolutionary Search Among Adversaries," Ph.D. dissertation, University of California, San Diego, San Diego, CA, 1997.

[12] Siang Yew Chong and Peter Tino and Xin Yao, "Measuring generalization performance in coevolutionary learning," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 4, pp. 479–505, 2008.

[13] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimisation procedure," *Artificial Life*, vol. 2, no. 10, pp. 313–323, 1991.

[14] T. Ranjeet, "Coevolutionary algorithms for the optimisation of strategies for red teaming applications," Ph.D. dissertation, School of Computer and Security Science, Edith Cowan University, 2012.

[15] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Prentice Hall International, 2014, ISBN13: 978-1292061429.

[16] D. Wiest, "Some properties of schedules for large projects with limited resource," *Operations Res.*, vol. 12, pp. 395–416, 1964.

[17] B. Gavish and H. Pirkul, "Algorithms for multi-resource generalized assignment problem," *Mgmt. Sci.*, vol. 37, no. 6, pp. 696–713, 1991.

[18] A. Moselhi and P. Lorterapong, "Least impact algorithm for resource allocation," *Can. J. Civ. Engineering*, vol. 20, no. 2, pp. 180–188, 1993.

[19] F. Talbot and J. Patterson, "Optimal methods for scheduling projects under resource constrains," *Proj. Mgmt. Quarterly*, vol. 12, no. 1, pp. 26–33, December 1979.

[20] E. W. Davis and J. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Mgmt. Sci.*, vol. 21, no. 8, pp. 944–955, 1975.

[21] D. Pham and D. Karaboga, *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer Verlag, 1999.

[22] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed., ser. Natural Computing. Springer Verlag, 2015, iSBN-13: 978-3662448731.

[23] K. Weicker, *Evolutionäre Algorithmen*, 3rd ed. Wiesbaden, Germany: Springer Vieweg, 2015, iSBN-13: 978-3658099572.