A Parallel Approach to Elevator Optimization Based on Soft Computing

Thomas Bartz–Beielstein* Sandor Markon† Mike Preuss*

*Universtität Dortmund
D-44221 Dortmund, Germany
{tom,preuss}@LS11.cs.uni-dortmund.de

†FUJITEC Co.Ltd. World Headquarters
28-10, Shoh 1-chome, Osaka, Japan
markon@rd.fujitec.co.jp

1 Introduction

Efficient elevator group control is a complex combinatorial optimization problem. Recent developments in this field include the use of reinforcement learning, fuzzy logic, neural networks and evolutionary algorithms [Mar95, CB98]. This paper summarizes the development of a parallel approach based on evolution strategies (ES) that is capable of optimizing the neuro-controller of an elevator group controller [SWW02]. It extends the architecture that was used for a simplified elevator group controller simulator [MAB+01, MN02, BEM03].

Meta-heuristics might be useful as quick development techniques to create a new generation of self-adaptive elevator group control systems that can handle high maximum traffic situations. Additionally, population based meta-heuristics such as evolution strategies can be easily parallelized. In the following we will consider a parallel elevator supervisory group control (ESGC) system that is based on a set of neural network-driven controllers, one per elevator shaft. These may be situated in one or several different buildings as long as communication between controller instances is enabled.

Since the ESGC problem is very costly in terms of computation time, a related dynamical system was introduced as simplified model: the sequential ring (S-ring) [MAB+01]. Using the S-ring also ensures that other researchers can compare their results with the ones presented here.

The rest of this article is organized as follows: in Sec. 2, we introduce one concrete variant of the elevator group control problem and show its relationship to the S-ring. The parallelization is described in Sec. 3, whereas Sec. 4 presents the experimental setup. Simulation results are analyzed in Sec. 5. The final Section combines a summary with an outlook.

2 The Elevator Supervisory Group Controller Problem

2.1 Problem Definition

The elevator ESGC problem subsumes the following problem: how to *assign elevators to passengers* in real-time while optimizing different elevator configurations with respect to overall service quality, traffic throughput, energy consumption etc. This problem is related to many other stochastic traffic control problems, especially with respect to the complex behavior and to many difficulties in analysis, design, simulation, and control [Bar86, MN02].

The ESGC problem considered here consists of a neural network (NN) controlling the behavior of elevator cars during simulation of a predefined traffic situation. Some of the NN connection weights can be modified, so that different weight settings and their influence on the ESGC performance can be tested. These are subject to optimization by means of an ES. Measuring performance of one specific weight setting by simulation automatically leads to stochastically disturbed (noisy) fitness function values.

2.2 A Simplified ESGC Model

We propose a simplified and easily reproducible ESGC model, the 'S-ring model'. Elevator cars in the S-ring model have unlimited capacity, and passengers are taken, but not discharged. The running directions of the cars are only reversed at terminal floors. All floors are indistinguishable: there are identical passenger arrival rates on every floor, and identical floor distances. The cars use uniform running and stopping times, and the whole model uses discrete time steps.

The system state at time t is given as

$$[s_0(t), c_0(t), \ldots, s_{n-1}(t), c_{n-1}(t)] \equiv \mathbf{x}(t) \in \mathbf{X} = \{0,1\}^{2n}.$$

There are n sites, each with a 2-bit state (s_i, c_i), and with periodic boundary conditions at the ends. s_i is set to 1 if a server is present on the ith floor, otherwise it is set to 0. The same applies to the c_i bits: they are set to 1 if at least one customer is waiting on the ith floor. Instead of using synchronous updating at all sites independently, one updating cycle is decomposed into n steps as follows: The state evaluation is sequential, scanning the sites from $n-1$ to 0, then again around from $n-1$ (see Fig. 1) [MAB$^+$01, BEM03]. At each time step, one triplet $\xi \equiv (c_i, s_i, s_{i+1})$ is updated, the updating being governed by the stochastic state transition rules, and by the policy $\pi : \mathbf{X} \to \{0,1\}$. A new customer arrives on the ith floor with probability p, and we consider m different elevator cars.

The S-ring can be used to define an optimal control problem, by equipping it with an objective function Q (here E is the expectation operator):

$$Q(n, m, p, \pi) = \mathrm{E}\left(\sum c_i\right). \tag{1}$$

Q can be read as the expected number of floors with waiting customers. For given parameters n, m, and p, the system evolution depends only on the policy π, thus this can be written as

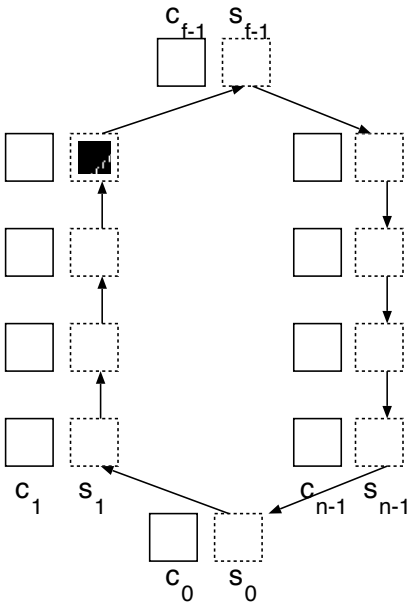

Figure 1: The S-ring as an elevator system

$Q = Q(\pi)$. The optimal policy is defined as

$$\pi^* = \arg\min_{\pi} Q(\pi). \tag{2}$$

The basic optimal control problem is to find π^* for given parameters n, m, and p.

The S-ring can be solved exactly for small problem sizes, while still exhibiting non-trivial dynamics.

3 A Parallel Approach

The previously introduced S-ring objective function (Eq. 1) describes the quality of a policy π for given parameters p, n, and m, thus for a fixed building and passenger arrival rate. In a more realistic setting, an ESGC has to cope with different passenger flow levels within each day. Furthermore, if the same ESGC is intended for use with multiple building configurations, it might be useful to have solutions for many floor/server number combinations at hand. This leads us to a policy table, with a p, n, and m setting as key and an optimized policy as value.

The motivation for computing such a table by means of a parallel architecture is twofold. Considering the offline case where control policies must be optimized before they are applied, the number of entries and thus the number of noisy optimal control problems to solve can be quite high. This results in a vast amount of required computation time which is reduced by parallelization.

On the other hand, a distributed optimization process may also be interpreted as a model for a network of controllers, each of them optimizing the policy for their current traffic situation

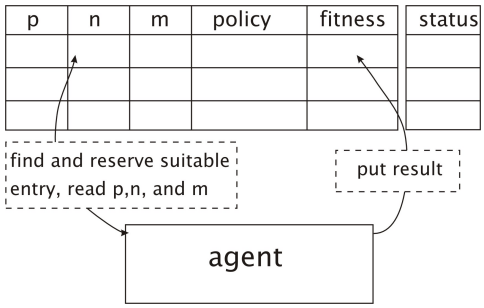

Figure 2: Agent-centered view of decentralized task assignment based on a policy table. Each agent locates an empty entry, marks it as reserved, computes the associated policy optimization task and stores the best found policy back to the table (see Sec. 3.2).

online. Controllers situated in different buildings or wings of the same building may profit from exchanging good policies with each other if these are applicable.

3.1 Successful Genes

The parallel architecture presented in the following is independent of the ESGC problem. It simply depends on the separability of the overall task which is the optimization of controller policies for the different policy table entries, each of these corresponding to a basic optimal control problem as described in Sec. 2.2. By modelling ESGC problem instances with an S-ring model we strive for keeping computational effort low and analysis as simple as possible. In the context of evolutionary algorithms, policies can be interpreted as *successful genes* that have to be combined to make up a complete genome.

Each table entry (or gene) consists of five columns. The numbers in the first column specify the arrival probabilities p, whereas the entries in the second and third column characterize the properties of a building which are the number of floors n and of elevator cars m, respectively. The policy π and the resulting objective value Q are given in column four and five.

It may be worthwhile to explore the possibility of policy exchange between different buildings. For identical n and m values, interchangeability is guaranteed. But even if these are different, policies may be transferred if buildings belong to the same equivalence class. For now, there are no concrete rules for deriving these, but we expect to get some hints from the experimental results. However, this implies solving the problem of conversion between NN weight configuration vectors $\vec{x}$ of different cardinalities.

3.2 Parallel Model of Computation

The parallel model of computation suggested here is suited for offline as well as online policy optimization. For the latter, independent computational units are mandatory because each of the controllers must remain fully operable even if all connections to other controllers are lost. In this respect, any centralized communication scheme would fail. Therefore, we assume

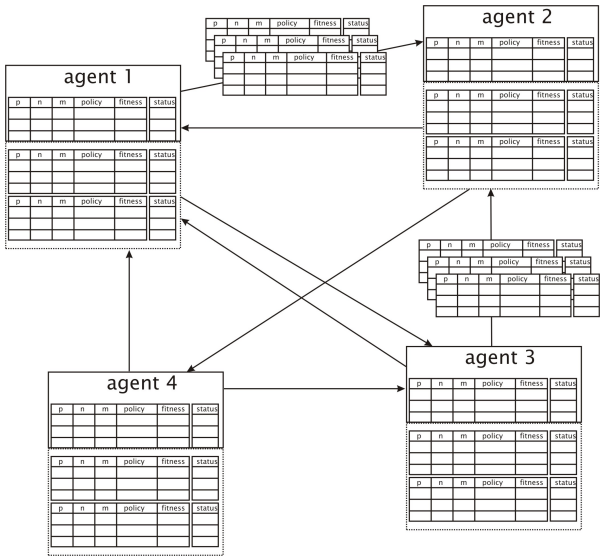

Figure 3: Exchange of tables between the agents. Note that each agent keeps its own and a number of remote agents tables. The set of agents known as well as the table set is not necessarily complete. At a regular time interval, each agent uses one of the connections indicated by an arrow to exchange the table sets (in progress for agents 3 and 2 and agents 1 and 2 here).

that a given number of distributed agents with the ability to communicate with each other can be employed for computing a policy table. In case of an online optimization, these may be embedded with the controllers of a network.

Every agent starts with an empty table of size l, where only n, m, and p are set for each entry. Additionaly, entries store a status which is one of: `empty`, `reserved`, or `finished` , all set to `empty` at the beginning. Then, the agents run the following algorithm (see Fig. 2):

1. determine entry i randomly
2. beginning with i, select the next `empty` entry
3. if no `empty` entry is found, select the next `reserved` entry, starting again with entry i
4. if no `reserved` entry is found, terminate
5. mark the selected entry as `reserved` and perform policy optimization on the associated n, m, and p setting
6. store the best found policy and mark the selected entry as `finished`
7. jump to step 1.

At the same time, agents exchange their tables assynchronously by use of the *Newscast Computing* scheme [JvS02]. Besides the local table, each agent also stores and communicates copies of the most recent known tables from some of the other agents, see Fig. 3. The recipients then use these to update the local table by importing entries marked as `reserved` or `finished`.

The suggested model of computation ensures robustness as well as scalability. The latter is guaranteed by a fixed upper limit of stored tables and connections within each agent. Thereby,

broadcasting is avoided at the cost of a logarithmic rather than constant update time order. The number of employed agents is only limited by the number of table entries. As policies for the entries remaining empty will be computed by the agents left, agent loss during runtime generally only slows down the process. There is however an inherent drawback that comes with decentralized task assignment: if two agents chose to work on the same task before they inform each other that it is reserved, some work is doubled. To minimize this effect, communication speed has to be chosen such that the average time needed to finish one task is much higher than the one used up for sending the latest results.

3.3 Implementation

The suggested parallel model of computation has been implemented based on the DREAM platform [ACE+02]. This is a peer-to-peer-based multiagent system primarily designed for, but not limited to, the run of parallel evolutionary algorithms.

4 Experiments

Experimental layout has been driven by two major aims. First, we need to make sure that using an evolution strategy for optimizing a single ESGC problem is successful so that it provides a NN weight setting $\vec{x}$ resulting in a better fitness $f(\vec{x})$ than assigned to the default weight setting. The latter uses 1.0 for each weight, corresponding to a controller behavior where every waiting customer is served.

Second, we want to test the efficiency of our parallelization model. Here, we face a difficulty introduced by the use of a peer-to-peer-based system: the underlying hardware is highly inhomogeneous, as opposed to more traditional parallel architectures where processing units reside in one physical machine or several identical machines are connected by a fast network. Neither speed nor network latency times of the nodes building the DREAM are equal. Furthermore, nodes can get temporarily disconnected or even switched off during runtime of the experiment. Thus, speedup or efficiency measuring requires some modifications to established definitions.

4.1 Modified Speedup Definition

A common definition [BH92] for speedup achieved by a parallel algorithm running on p processors is:

$$S(p) = \frac{\text{Time needed with the fastest serial code on a specific parallel computer}}{\text{Time needed with the parallel code using } p \text{ processors on the same computer}} \quad . \quad (3)$$

Efficiency, defined as the fraction of linear speedup attained, is then $E(p) = S/p$. As the number of processors employed does not allow for rating the computational power available in an inhomogeneous parallel architecture, we measure this resource by applying a Linpack benchmark on each processing node. Thus, p in (3) can be resembled by the relative power $\hat{p}_{rel}$ according to the following definition:

$$\hat{p}_{rel} = \frac{\sum_n \hat{p}_i}{\hat{p}_{single}} \tag{4}$$

Here, n denotes the number of nodes available, $\hat{p}_i$ is the measured computing power of one node, and $\hat{p}_{single}$ is the measured power of the node which performed the sequential run. Note that the problem dealt with is not optimization of one policy but of a whole policy table. The fastest serial code solving this problem therefore simply implements a loop over all entries of the table without any communication. As we try to determine the speedup of our parallelization model rather than of the utilized evolutionary algorithm, speedup considerations concerning parallel evolutionary algorithms as in [AT02] are not applicable here.

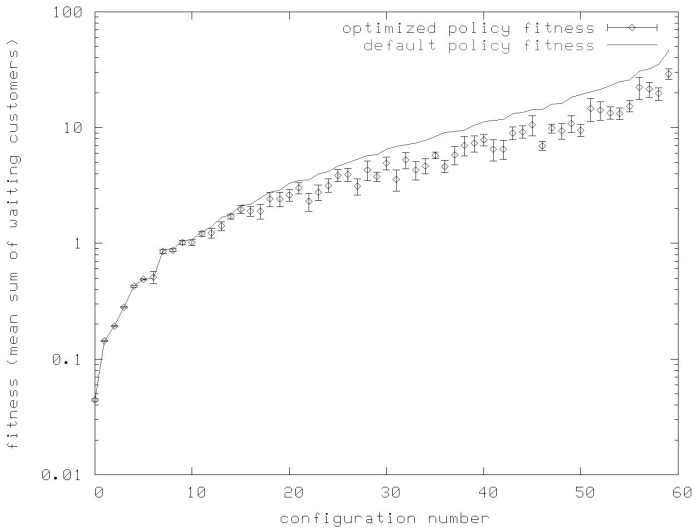

Figure 4: Fitness of optimized policies compared to the default policy fitness for 60 different settings of p, n, and m, representing all entries of a policy table. The configurations are orderered by the default policies quality, thus from rather simple to harder optimization problems. Note that the objective function employed here describes a minimization problem by definition. Presented numbers are averages over 25 runs, error bars represent the standard deviation.

4.2 ES settings

For optimization of the policy associated with each policy table entry, a (10+50) evolution strategy with fixed number of evaluations is employed. Due to the stochastically disturbed nature of our objective function, higher selection pressures seem to be counterproductive because they tend to prefer outliers produced by noise rather than by good policies.

Each objective function evaluation requires simulating 1000 iterations of the appropriate S-ring model. To allow for enhanced analysis possibilities, we use a slightly modified objective function which results in counting the total number of customers waiting as opposed to the number of floors with waiting customers. Extensive testing shows that the standard deviation

of the noise generating process contributing to the resulting fitness is quite high, around 2.5%. Increasing the number of simulation steps lowers this level but also increases the effort needed dramatically: for 100,000 steps, the average inaccuracy due to noise drops down to approximately 0.3%. We therefore chose to reevaluate the best policy found with the latter setting while the former is used for fitness evaluation during optimization runs.

Several methods exist to increase the performance of evolution strategies on noisy objective functions, one of which is threshold selection [MAB+01]. Another one employed here consists of reevaluating surviving parents each generation and thus accumulating a moving average fitness value for the long-lived individuals. Thereby, we hope to prevent extreme outliers produced by the overlaying noise from dominating the population for too long.

4.3 Experiment layout

In order to simplify comparison of results, policy tables have been chosen such that they contain multiples of 20 entries. The set of floor numbers n used is { 6, 8, 10, 12, 14 }, the set of elevator car numbers m is { 2, 3, 4, 5 }. According to the desired table size, we apply one to three arrival probabilities p from the set { 0.1, 0.2, 0.3 }.

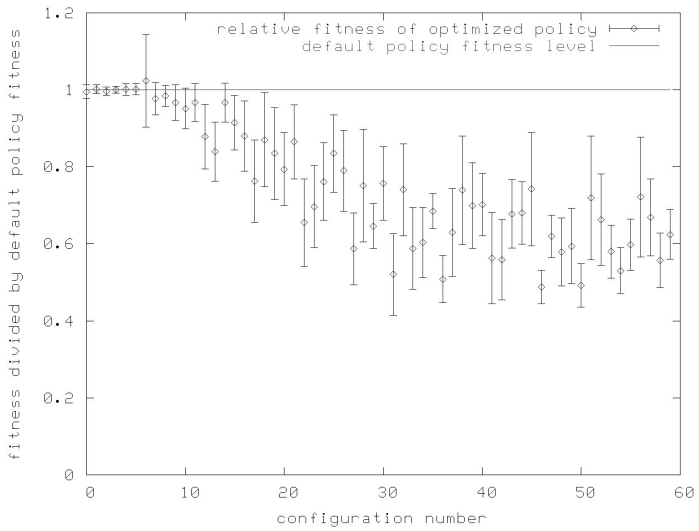

Figure 5: Same data as in figure 4, here optimized policy fitness values are divided by the default policy fitness value of the same configuration. This view shows the distribution of the optimization potential easily exploited.

5 Evaluation

We now separately evaluate the quality of policy optimization by means of an evolution strategy and the degree of parallelization for optimizing a whole policy table achieved by the model of computation outlined in Sec. 3.

Table 1: Optimization results for $n = 10$ (number of floors), subset of the results depicted in figure 4. Values averaged from 25 runs. Columns p and m denote arrival rate and number of elevators, respectively.

p	m	fitness	std.dev.	default fitness	std.dev.	$\frac{\text{fitness}}{\text{default fitness}}$	gain in %
0.1	2	4.3184	0.8395	5.7441	0.0310	0.7518	24.8202
0.1	3	2.7730	0.4225	3.9795	0.0139	0.6968	30.3178
0.1	4	1.8921	0.2660	2.4791	0.0170	0.7632	23.6796
0.1	5	1.2229	0.1172	1.3916	0.0108	0.8788	12.1238
0.2	2	9.0038	1.1861	13.2834	0.0397	0.6778	32.2175
0.2	3	5.8251	1.0568	9.2437	0.0482	0.6302	36.9829
0.2	4	3.7591	0.3431	5.8183	0.0288	0.6461	35.3909
0.2	5	2.6162	0.3140	3.2956	0.0234	0.7938	20.6168
0.3	2	14.6404	3.2640	20.3561	0.0857	0.7192	28.0787
0.3	3	9.2216	1.1026	13.5584	0.0825	0.6801	31.9863
0.3	4	5.7845	0.3906	8.4397	0.0441	0.6854	31.4606
0.3	5	3.9424	0.5257	4.9908	0.0278	0.7899	21.0081

5.1 Optimization Quality

Although we allowed only 10000 objective function evaluations for each run, Fig. 4 indicates that most of the optimized policies perform considerably better on the S-ring model than the default (always stop) policies. Especially in situations with higher throughput per elevator car when the default policy produces greater numbers of waiting customers, optimized policies offer a valuable alternative. Table 1 provides numerical results for a subset of the 60 policies.

In Fig. 5, we provide an overview of the relative fitness decrease which illustrates a high result diversity for one parameter setting as well as for configurations regarded as similarly difficult. We believe that this points to a still unexploited optimization potential and thus some room for improvement of the evolution strategies performance.

However, it must be stated that arrival probabilities p for all 60 configurations have been rather low at 0.1 to 0.3. Separate test runs with higher values of p exhibited increasing loss of optimization performance. We thus expect to find the best optimization results for medium throughput situations.

Table 2: Speedup comparison for policy tables of size 20; averages from 10 runs.

efficiency E	speedup S	nodes n	relative power p_{rel}	runtime in s	$\sum_n \hat{p}_i$ in MFlops
1.0	1.0	1	1.0	4450.5304	59.9596
0.797	4.048	5	5.076	1099.4618	304.3315
0.491	4.827	10	9.823	922.0467	588.9555

Table 3: Speedup comparison for policy tables of size 40; averages from 10 runs.

efficiency E	speedup S	nodes n	relative power p_{rel}	runtime in s	$\sum_n \hat{p}_i$ in MFlops
1.0	1.0	1	1.0	8659.2744	59.4248
0.8	4.716	5	5.895	1833.9098	350.3132
0.644	6.952	10	10.800	1245.5769	641.7714

Table 4: Speedup comparison for policy tables of size 60; averages from 10 runs.

efficiency E	speedup S	nodes	relative power p_{rel}	runtime in s	$\sum_n \hat{p}_i$ in MFlops
1.0	1.0	1	1.0	1.2571E+04	58.4689
0.724	5.463	5	7.547	2301.2433	441.2680
0.653	6.596	10	10.095	1905.7683	590.2685

5.2 Speedup Considerations

Unlike most other task distribution schemes, our parallelization model provides each processing node with a full set of task results and is therefore suitable to the online case where policy optimization is performed during controlling time without modification. On the other hand, efficiency may be lower than can be expected for centralized schemes like server-client models.

Results for policy table sizes of 20, 40 and 60 entries are presented in tables 2, 3, and 4. The runtime column shows that computing time for a whole table has been considerably reduced in the parallel case, leading to an efficiency up to 80% for 5 processing nodes. As the number of nodes grows towards the number of table entries, efficiency decreases because entry selection conflicts resulting in doubled work become more likely. It must be noted that using bigger tables would have been desirable but also increases the runtime for the sequential run dramatically.

6 Summary and Outlook

We introduced a parallel approach for a distributed ESGC problem. Due to the separability of the objective function an efficient evaluation scheme could be implemented (successful genes). Experiments performed confirm the applicability of this approach, proving efficiency of the parallelization as well as significant quality gains due to optimization by means of an evolutionary algorithm. The employed task distribution model is well suited to the offline case where a policy set is fully optimized prior to its implementation and looks promising concerning the online real-time optimization case. In future, the latter variant deserves some more attention.

The current work can be extended by introducing car capacities, or implementing different strategies that are working in parallel. Additionally, this methodology is transferable to other traffic systems or, more general, other distributed control problems.

Acknowledgments. This research was supported by the DFG as a part of the collaborative research center 'Computational Intelligence' (531).

References

[ACE+02] M. G. Arenas, P. Collet, A. E. Eiben, M. Jelasity, J. J. Merelo, B. Paechter, M. Preuß, and M. Schoenauer. A framework for distributed evolutionary algorithms. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J. L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VII, Proc. Seventh Int'l Conf., Granada, September 2002*, volume 2439 of *Lecture Notes in Computer Science*, pages 665–675, Berlin, 2002. Springer.

[AT02] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[Bar86] G. Barney. *Elevator Traffic Analysis, Design and Control*. Cambridg U.P., 1986.

[BEM03] T. Beielstein, C. P. Ewald, and S. Markon. Optimal elevator group control by evolution strategies. In *Proc. 2003 Genetic and Evolutionary Computation Conf. (GECCO'03), Chicago*, Berlin, 2003. Springer.

[BH92] R. Barr and B. Hickman. Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. *ORSA Journal on Computing*, 1992.

[CB98] R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.

[JvS02] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, October 2002.

[MAB+01] S. Markon, D. V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer. Thresholding – a selection operator for noisy ES. In J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu, editors, *Proc. 2001 Congress on Evolutionary Computation (CEC'01)*, pages 465–472, Seoul, Korea, May 27–30, 2001. IEEE Press, Piscataway NJ.

[Mar95] S. Markon. *Studies on Applications of Neural Networks in the Elevator System*. PhD thesis, Kyoto University, 1995.

[MN02] S. Markon and Y. Nishikawa. On the analysis and optimization of dynamic cellular automata with application to elevator control. In *The 10th Japanese-German Seminar, Nonlinear Problems in Dynamical Systems, Theory and Applications*. Noto Royal Hotel, Hakui, Ishikawa, Japan, September 2002.

[SWW02] H.-P. Schwefel, I. Wegener, and K. Weinert, editors. *Advances in Computational Intelligence – Theory and Practice*. Natural Computing Series. Springer, Berlin, 2002.