Solving DCOPs in Self-optimising Multi-Agent Systems by Extending the Local Objective Functions

Sebastian Niemann and Christian Müller-Schloer
Institute of Systems Engineering, System and Computer Architecture,
Leibniz Universität Hannover, Hannover, Germany,
{niemann,cms}@sra.uni-hannover.de

Abstract

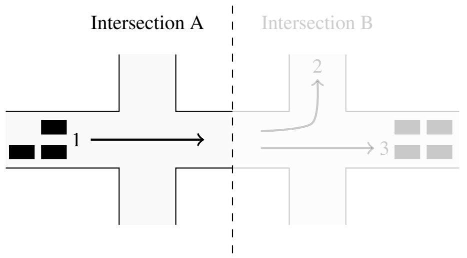
Several applications of Organic Computing (OC) systems as well as Autonomic Computing (AC) systems are based on self-optimising multi-agent systems, i.e. distributed autonomous devices. One of the main challenges of these is to emerge towards a global optimal system state based only on local information for each agent. In order to reach a global optimal state some agents need to avoid selfish actions and instead consider the benefits of their actions for the whole system. Choosing the action of an agent is often based on solving optimisation problems, which can be modelled as a distributed constraint optimisation problem (DCOP). This paper presents a new asynchronous approach to solve DCOP by extending only the underlying local objective function of each agent. The main benefit of this approach is the avoidance of an additional complex decision making algorithm that may interfere with the original task of an agent and reduces the scalability of the system. Exemplary, a distributed constraint optimisation problem is considered to quantify the effectiveness and computation as well as communication cost of the discussed approach.

1 Introduction

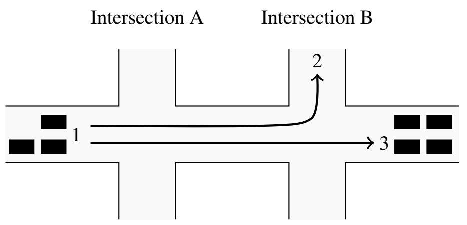
Self-organising systems within Organic Computing (OC) as well as Autonomic Computing (AC) often consist of multi-agent systems, i.e. distributed computing devices, which only interact based on local decisions among themselves. A large class of self-organising systems are actually based on self-optimising systems, where each device try to solve a local optimisation problem [10]. Self-optimising systems have been applied in several fields, such as traffic networks [12, 15], communication networks [6] and wireless sensor networks [13]. In order to find the best set of actions for a distributed self-optimising system at least some agents need to avoid *selfish* optimisation tasks where it did not consider its influence on other agents during the optimisation process. For example, in order to reduce the overall workload of a computer cluster, in general the most suited, e.g. powerful node needs to accept more work than any other node instead of reducing its local workload by declining all work request.

The optimisation process within self-optimising systems can be divided into reaching an optimal solution and maintaining it during runtime. The task of reaching an optimal solution requires to cover a large part of the search space, if not the whole search space, and therefore may require a lot of computation time. Besides the initialisation step, it may also be necessary to redetermine, i.e. reach again an optimal solution after fundamentally changes within the optimisation problem. In general, maintaining the solution requires a much lesser computation time in comparison to reaching it, as it only adjusts a yet optimal solution in occurrence of small alterations within the optimisation problem. However, the acceptable computation time for determining an optimal solution during runtime is usually also significant lower compared to the computation time restrictions during design time. Therefore, redetermining an optimal solution during runtime without valid assumptions based on previous optimisation steps is the most crucial part of the optimisation process within self-optimising system. This paper also focus on reaching an optimal solution.

Agents within OC systems influence another by local actions which are mostly based on solving local optimisation problems. In order to reach the global optimal solution, the mutual influence between agents need to be taken into account. Consider the synchronisation of traffic lights as a self-optimising process [7]. It assumes that the traffic flow is completely controlled by the phase plan of intersections. The goal of this task is to reduce the overall waiting time of the vehicles within the system. Therefore, actions that overload an intersection need to be avoided. Figure 1 shows a partial view of a traffic system consisting of two intersections in which one needs to decide if the traffic should flow from intersection A to B, based on the local information of intersection A. Assume that intersection A should only allow the traffic to flow towards B, if this intersection directs the traffic towards B. Without including information about the influence on intersection B, it is not possible to conclude if allowing vehicles to travel to intersection B will result in an overload of the system, i.e. intersection B (see Figure 1a). However, increasing the horizon about the influence on neighbouring intersections as shown in Figure 1b allows to conclude what the action has the higher benefits. However, without synchronisation between agents, an uncertainty about the correctness of the assumed influence remains.



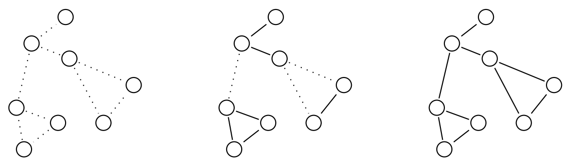
(a) Without consideration on the influence between the intersections



(b) With consideration on the influence between the intersections

Figure 1: Partial view of a traffic system

We differentiate between three classes, e.g. states of optimisation problems, based on the given information about the influence on neighbouring agents. An optimisation problem is thereby either in a *a*) loosely, *b*) partially connected or *c*) (fully) connected state (see Figure 2). Within the loosely state, the local informations of an agent did not provide information about the influence on neighbouring agents, while in the partially connected states, subgroups of agents can conclude their mutually interference. After reaching the (fully) connected state, the system provides complete information of the influence between agents. Based on this information, an optimisation problem can be modelled as a distributed constraint optimisation problem (DCOP) [3]. This paper focus on such problems and therefore requires that a (fully) connected state was reached a priori to the proposed approach.



(a) Loosely (b) Partially connected (c) Connected

Figure 2: Differentiation between states of optimisation problems based on information about the influence on neighbouring agents. Dotted lines denote interactions without information on their influence, in contrast to solid lines

A DCOP can be modelled as a graph, where each agent is represented by a node and each edge denotes a soft constraint between agents, such that the most desirable joint action is favoured. These soft constraint are equivalent to the evaluation of the benefit of an agent based on the influence with another agent. Later in this paper the concept of soft constraint will be formally introduced.

1.1 Related Work

DCOP algorithms can be divided into *complete* and *incomplete* algorithms. Complete DCOP algorithms are guaranteed to find global solutions while incomplete algorithms may settle with a local solution. However, incomplete DCOP algorithms often achieve satisfying solution with a much lower computation and communication cost [17].

Complete DCOP algorithms such as Optimal Asynchronous Partial Overlay (OptAPO) [8] consist of at least one central agent that collects information on a subset of other agents and guide the optimisation process. DCOP algorithms without central agents like Asynchronous Distributed optimisation (ADOPT) [9], Branch-and-Bound ADOPT (BnB-ADOPT) [17] and No-Commitment Branch and Bound (NCBB) [1] arrange the agents into a tree initially and make either use of backtracking or branch and bound. The tree ordering allows to compute sibling sub-trees independently during the optimisation process. Recent interest on incomplete DCOP algorithms is focused around *k*-optimal algorithms which guarantee to find solutions that cannot be improved by changing only *k* agents at most [11]. However, DCOP algorithms are limited to optimisation problems with finite domains in order to guarantee to find a global or at least *k*-optimal solution in a reasonable amount of time without further information on the problem, such as derivations and convexity.

1.2 Main idea and structure of the paper

This paper presents an approach to optimise distributed optimisation problems by extending the local objective function of an agent. The extended objective function works by adding the current objective value of neighbouring agents upon its own. The objective value of neighbouring agents are thereby be shared every time their own action changes. The main benefit of this approach is the avoidance of an additional complex decision making algorithm. This approach can be used with continuous domains in contrast to the discussed DCOP algorithms. However, note that this paper did not provide an optimisation algorithm itself but rather an extension on the local objective functions.

Sect. 2 briefly reviews the fundamentals of distributed constraint optimisation problems. In sect. 3 the proposed approach is elucidated. Sect. 4 demonstrates the capability of the proposed optimisation procedure. Exemplary, a distributed constraint optimisation problems is considered in order to quantify the effectiveness of the proposed approach. After a brief description of the optimisation problem, the results of the optimisation procedure are presented and discussed. Sect. 5 concludes this paper and discusses future work.

2 Optimisation Problem

An optimisation problem is the problem of finding the best (acceptable) solution for a given objective function within a reasonable amount of time. Formally, an optimisation problem is defined by an objective function $F : D \to \mathbb{R}$, the search space $S \subseteq D$ and a goal. D is the domain of the function F. The search space S is the space of all acceptable elements of the domain. An element $x \in S$ of the search space is called a solution. Commonly, the goal is either to find the minimal or maximal value of the objective function within the search space. By convention, the optimisation problem is viewed as a minimization problem.

$$\underset{x\in S}{\text{minimize}}\, F(x),\ F : S \to \mathbb{R}. \tag{1}$$

2.1 Distributed Constraint optimisation Problem

In a distributed constraint optimisation problem the solution x is separated in local solutions x_i and the objective function and a set of soft-constraints $f_{i,j}$ between these partial solution are introduced. Formally, the domain D and the objective function F is separated into a finite amount of sub-domains $D_i \subseteq D,\ i \in N,\ 1 < |N| < \infty$, and local objective functions $f_i : D_i \to \mathbb{R},\ i \in N$, respectively. N is the index set of the partitioning. The search space and the solution of a sub-function f_i is denoted as S_i, and $x_i \in S_i$, respectively. The solution x is denoted as:

$$x = \{x_0, \ldots, x_{|N|}\} \tag{2}$$

A soft-constraint $f_{i,j} : D_i \times D_j \to \mathbb{R},\ i \neq j$ maps every combination of partial solutions x_i, x_j to an objective value. The local objective functions f_i is obtained by summation over the corresponding soft-constraint.

$$f_i(x_i) = \sum_{j\in N\setminus\{i\}} f_{i,j}(x_i, x_j) \tag{3}$$

Since DCOP is based on local information, the local objective functions f_i actually excludes global information within the calculation. We will therefore redefine the local objective functions f_i in a more complex way, based on local information only. Each agent i defines a so called context C_i. A context C_i of agent i is a set of pairs (j, x_j) that contain the index j and the local solutions x_j of agent j based on the local information of the i-th agent. Note that the context C_i does not necessarily includes the actual solution x_j of agent j. It may also include no pair (j, x_j) whether or not there is a soft-constraint between agent i and j. This incompleteness of the context is also the main reason why the index j is included. The local objective functions f_i is than redefined as:

$$f_i(x_i) = \sum_{(j,x_j)\in C_i} f_{i,j}(x_i, x_j) \tag{4}$$

The function that combines the sub-functions in order to get the objective function F is denoted as $\Theta : \mathbb{R}^N \to \mathbb{R}$.

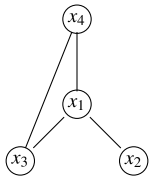
$$F(x) = \Theta(f_1(x_1), \ldots, f_{|N|}(x_{|N|})) \tag{5}$$

We assume that the local objective functions f_i and the (global) objective function F share a common goal. Thus, the optimisation of a local objective function f_i should also optimise the objective function F. Therefore, we assume that the local objective functions can be optimised independent of Θ.

$$F\left(\underset{x\in S}{\arg\min}\begin{pmatrix} f_i(x_i) \\ \vdots \\ f_i(x_{|N|}) \end{pmatrix}\right) = \min_{x\in S} F(x) \tag{6}$$

Note that this did not implies that the local objective function can be optimised independently from another. In most recently discussed cases, Θ is assumed to be the sum over all local objective values [16]. For simplification, Θ is also assumed to be the sum over all local objective values within this paper for all given examples.

Figure 3 shows an example of a DCOP represented by a (soft-)constraint graph and a local objective function that shall be equal for all agent i. Each agent is represented by a circle and chooses only one partial solution $x_i \in \{0, 1\}$. The global optimal solution of this example would be $x = (1, 1, 1, 1)$ with $F(x) = 0$.



(a) Soft-constraint graph

x_i	x_j	$f_{i,j}(x_i,x_j)$
0	0	1
0	1	2
1	0	2
1	1	0

(b) Local objective function

Figure 3: Example of a DCOP, based on the work of Modi et al. [9]

3 Details of the Approach

The key idea behind this approach is to extend the existing local objective function, such that its optimal solution represents the most beneficial local action for the whole system. Therefore, each agent includes within the calculation of its local objective function f_i, how a solution x_i affects the sub-function f_j of other agents $j \in \mathcal{N} \setminus \{i\}$. Instead of considering all other agents of the whole system, we only include the influence on neighbouring agents. An agent j is considered to be an neighbour of agent i if they share a soft-constraint. The set of all neighbours of agent i is denoted as Ω_i.

$$\Omega_i := \{j \in N \setminus \{i\} : \exists f_{i,j}\} \tag{7}$$

$$\exists f_{i,j} \Leftrightarrow \exists (x_i,x_j) \in S_i \times S_j : f_{i,j}(x_i,x_j) \neq 0 \tag{8}$$

The affect of a solution towards neighbouring agents is measured by the soft-constraints of a given DCOP. After choosing an action x, the current objective value $f_i(x)$ is transferred to all local objective functions of neighbouring agents $j \in \Omega_i$ (see Figure 4). Note that whenever an agent chooses another solution x_i, all neighbouring agents $j \in \Omega_i$ need to be informed in order to update their context C_j.

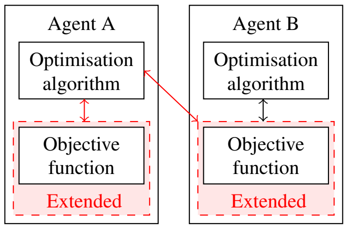


Figure 4: Concept of the proposed approach

After receiving an objective value $f_j^r(x_i) = f_j(x_j)$, it is saved together with the last action x_i that was send to the agent j. The extended local objective function f_i^e of an agent i is then calculated by the summation of the original objective function f_i and all received objective values of f_j^r from neighbouring agents.

$$f_i^e(x_i) = f_i(x_i) + \sum_{j \in N \setminus \{i\}} f_j^r(x_i) \tag{9}$$

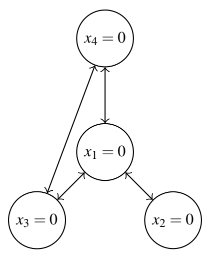
The following pseudo-code provides information about the implementation of the presented approach. All

Algorithm 1: Basic implementation

```
// Initialisation
$x_i = Random$
foreach $j \in \Omega_i$ do
    Send $x_i$ to agent $j$
    Receive $x_j$ from agent $j$
end
Update $f_i$
// Barrier
foreach $j \in \Omega_i$ do
    Send $f_i(x_i)$ to agent $j$
    Receive $f_j(x_j)$ from agent $j$
end
Update $f_i$
// Start
repeat
    if $(x_j^{next}, f_j(x_j^{next}))$ received from agent $j$ then
        Update $f_i^e$
    end
    // Optimisation algorithm
    $x_i^{\text{next}} = \arg\min_{\tilde{x}_i \in S_i} f_i^e(\tilde{x}_i)$
    if $x_i \neq x_i^{next}$ or $f_i^e(x_i) \neq f_i^e(x_i^{next})$ then
        foreach $j \in \Omega_i$ do
            Send $(x_i^{\text{next}}, f_i(x_i^{\text{next}}))$ to agent $j$
        end
    end
    $x_i = x_i^{\text{next}}$
until termination
```

agents choose an initial solution randomly and send these to all neighbouring agents in order to populate the context C_i of each agent. After receiving all solutions x_j from neighbours $j \in \Omega_i$, each agents calculates their objective value and sends these towards all neighbouring agents. After this step is completed, each agent updates the local objective value in case new information about other agents is received. Based on the updated objective function and a suited optimisation algorithm, the next solution is obtained. Should neither the objective value nor the solution have changed, the solution is assumed to be static and no information is send to neighbouring agents, reducing the communication cost. If either the objective value or the solution have changed, all neighbouring agents are informed.

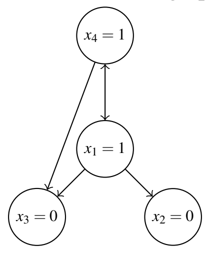
Consider the example given in Figure 3. Figure 5 demonstrates the process of the approach. Note that this is an asynchronous approach and especially non-deterministic.



(a) Soft-constraint graph

	$x_i = 0$	$x_i = 1$
$f_1(x_1)$	3 + 5	6
$f_2(x_2)$	1 + 3	2
$f_3(x_3)$	2 + 5	4
$f_4(x_4)$	2 + 5	4

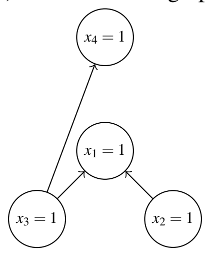
(b) Local objective function



(c) Soft-constraint graph

	$x_i = 0$	$x_i = 1$
$f_1(x_1)$	4 + 5	4
$f_2(x_2)$	2 + 3	0
$f_3(x_3)$	4 + 5	0
$f_4(x_4)$	3 + 5	2

(d) Local objective function



(e) Soft-constraint graph

	$x_i = 0$	$x_i = 1$
$f_1(x_1)$	6 + 5	0 + 0
$f_2(x_2)$	2 + 3	0
$f_3(x_3)$	4 + 5	0
$f_4(x_4)$	4 + 5	0 + 0

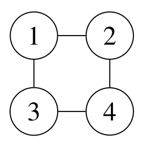
(f) Local objective function

Figure 5: Example of a the approach based on Figure 3

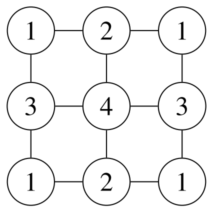
4 Example and Evaluation

In order to quantify the effectiveness of the discussed procedure, an grid based constraint graph with an binary domains $D_i = S_i$ is considered (see Figure 6).

$$D_i = S_i := \{0,1\},\ \forall i \in N \tag{10}$$



(a) 2x2 grid



(b) 3x3 grid

Figure 6: Possible interactions between each agent for the given example. The number in the middle describe the type of the agent

The grid contains four types of agents. Each type of an agent defines a different local objective function, which are used to avoid a global optimal solutions that can be obtained by ignoring the influence between agents. The type t of each agent is given within the circle in Figure 6. Based on this, the soft-constraints $f^t_{i,j}$ between the agents are defines as:

$$\begin{aligned} f^t_{i,j}(0,0) &= (0+t) \mod 4 \\ f^t_{i,j}(0,1) &= (1+t) \mod 4 \\ f^t_{i,j}(1,0) &= (2+t) \mod 4 \\ f^t_{i,j}(1,1) &= (3+t) \mod 4 \end{aligned} \tag{11}$$

The proposed approach is evaluated in comparison to the complete DCOP algorithm ADOPT. The variable-ordering a priori to ADOPT was done by the max-degree depth first search approach [5]. We implemented both ADOPT and the proposed approach based on Armadillo C++ [14] and OpenMPI-1 [4] in order to run each process asynchronously. The experiment was repeated 20 times for each grid. Since the proposed approach did not terminate, the optimisation was interrupted after 50ms and the state of the agents within this moment are presented in the following as the obtained solution.

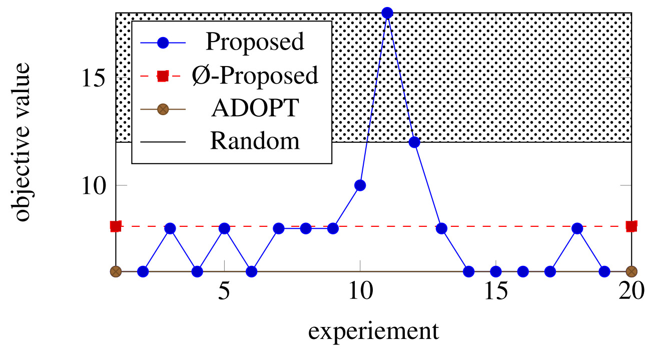
Figure 7 compares the reached objective value of the proposed approach against ADOPT. Note that ADOPT is a complete DCOP algorithm as stated before and therefore always finds the global optimal solution. We use this as an lower bound for the objective value. By calculating the average solution in case of random optimisation, we also get an upper bound for any *useful* approach [2].

The results demonstrate that the achieved solutions are nearly optimal in case of both an 2x2 and 3x3 grid. Especially the obtained results for the 3x3 grid are very promising. The optimal solution for the 2x2 grid is $x = (0,0,1,0)$ with an objective value $F(x) = 6$. The upper bound of the objective value based on random optimisation is 12. In this case the presented approach reached an average objective value of $8,1$. In case of the 3x3 grid the lowest objective value is 16 and the upper bound based on random optimisation 35. In this scenario, the presented approach reached an average objective value of $19,1$.

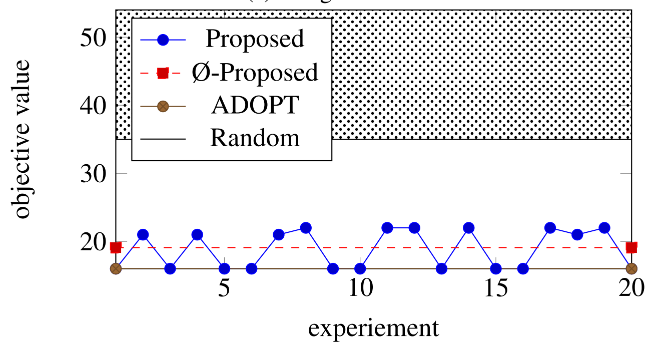
Besides these very promising results, the proposed approach reaches the worst objective value of 18 in the 2x2 grid case. This outlier can be explained by the sudden termination after 50ms. Further analyses of this case have shown that the local information of each agent directed the solution soon after this instant below an objective value of 10.

5 Conclusion and Challenges

In this paper, a new asynchronous approach to solve distributed constraint optimisation problems in self-optimising multi-agent systems was presented. After a



(a) 2x2 grid



(b) 3x3 grid

Figure 7: Results of the evaluation

short description of the main idea including a definition of the given optimisation problems, the proposed procedure was explained in detail. Basically, it integrates the influence on neighbouring agents into the the subproblem of an agent. Exemplary, a DCOP was considered to quantify the effectiveness of the procedure. The results demonstrate that the proposed approach is able reach a near optimal solution without the implementation of an additional complex decision making problem. Considering this paper as a first step towards a new asynchronous approach to solve distributed constraint optimisation problems in self-optimising multi-agent systems, the results are very promising.

While the results are very promising, we are currently missing a formal proof about reaching local optimal of our approach. Since this depends on the implementation of a termination process, we will focus on stabilising the local objective functions in order to reach a meaningful termination criterion. Future work will also deal with analysing the potential of combining classical optimisation algorithms for DCOPs with extending the objective function.

References

[1] Chechetka, A., and Sycara, K. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems* (2006).

[2] Dembski, W. *No Free Lunch: Why Specified Complexity Cannot Be Purchased Without Intelligence*. Rowman & Littlefield Pub Incorporated, 2007.

[3] Faltings, B. Distributed constraint programming. In *Handbook of Constraint Programming*. Elsevier, 2006.

[4] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting* (2004).

[5] Hamadi, Y., Bessière, C., and Quinqueton, J. Backtracking in distributed constraint networks. In *International Journal on Artificial Intelligence Tools* (1998).

[6] Hurling, B., Tomforde, S., and Hähner, J. Organic network control. In *Organic Computing*. Birkhäuser, 2011.

[7] Junges, R., and Bazzan, A. L. C. Modelling synchronization of traffic lights as a DCOP. In *Proceedings of the 5th European Workshop on Multiagent Systems* (2007), pp. 564–579.

[8] Mailler, R., and Lesser, V. R. Asynchronous partial overlay: a new algorithm for solving distributed constraint satisfaction problems. *Journal Of Artificial Intelligence Research 25* (2006), 529–576.

[9] Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence - Special issue: Distributed constraint satisfaction 161* (2005), 149–180.

[10] Müller-Schloer, C., Schmeck, H., and Ungerer, T., Eds. *Organic Computing - A Paradigm Shift for Complex Systems*. Birkhäuser / Springer, 2011.

[11] Pearce, J. P., and Tambe, M. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th international joint conference on Artifical intelligence* (2007).

[12] Prothmann, H., Tomforde, S., Branke, J., Hähner, J., Müller-Schloer, C., and Schmeck, H. Organic traffic control. In *Organic Computing*. Birkhäuser, 2011.

[13] Salzmann, J., Behnke, R., and Timmermann, D. Oc principles in wireless sensor networks. In *Organic Computing*. Springer Basel, 2011.

[14] Sanderson, C. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Tech. rep., NICTA, 2010.

[15] Tomforde, S. *An Architectural Framework for Self-configuration and Self-improvement at Runtime*. PhD thesis, Leibniz Universität Hannover, 2011.

[16] Yeoh, W., Felner, A., and Koenig, S. Idb-adopt: A depth-first search dcop algorithm. In *Recent Advances in Constraints, 13th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming* (2008).

[17] Yeoh, W., Felner, A., and Koenig, S. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal Of Artificial Intelligence Research 38* (2010), 85–133.