Swarm Intelligence Heuristics for Component Deployment

Máté J. Csorba, Poul E. Heegaard

Swarm Intelligence Heuristics for Component Deployment

Máté J. Csorba and Poul E. Heegaard

Department of Telematics,
Norwegian University of Science and Technology, N-7491 Trondheim, Norway
{Mate.Csorba, Poul.Heegaard}@item.ntnu.no

Abstract. We address the problem of efficient deployment of software services into a networked environment. Services are considered that are provided by collaborating components. The problem of obtaining efficient mappings for components to host in a network is challenged by multiple dimensions of quality of service requirements. In this paper we consider execution costs for components and communication costs for the collaborations between them. Our proposed solution to the deployment problem is a nature inspired distributed heuristic algorithm that we apply from the service provider's perspective. We present simulation results for different example scenarios and present an integer linear program to validate the results obtained by simulation of our algorithm.

1 Introduction

Implementing distributed networked software systems requires many important design decisions to be made that have strong influence on the Quality of Service (QoS) perceived by the user of the service. A major factor in satisfying QoS requirements of a software service is the configuration of the elementary building-blocks of the service and their mapping to the suitable network elements and resources that are available during execution. Moreover, it is also important for next generation software systems to be capable to (self-)adapt to foreseen and unforseen changes that appear in the execution context. This dynamism in the context of services is even increased by enabling swiftly reconfigurable hardware, allowing mobility and changes in the cardinality of users. Simply improving the QoS metrics without planing is, however, inevitably increasing the costs on the providers' side leading to multifaceted optimization problems.

We address the issue of obtaining efficient and adaptable mappings for software components of networked services as an optimization problem in a distributed environment. We model services as being built by collaborating components with several dimensions of QoS requirements including but not restricted to dependability, performance or energy saving aspects. Correspondingly, our service models are extended with costs relevant for the various dimensions of requirements in a more detailed model. Based on the service models we apply heuristics and a nature-inspired optimization method, called the Cross Entropy Ant System (CEAS) [1], [2], [3] to solve the problem of deploying service components into the network.

Distributed execution of our deployment mapping algorithm has been an important design criteria to avoid the deficiencies of existing centralized algorithms, e.g. performance bottlenecks and single points of failure. Moreover, we intend to conserve

resources by eliminating the need for centralized decision-making and the required updates and synchronization mechanisms. In our earlier work [4] we selected a well-known example in the domain of task assignment problems and converted it to our context of collaborating components with execution and communication costs. In this paper we extend the initial example from [4] with two additional example system models, present an Integer Linear Program (ILP) able to solve component mapping problems with load-balancing and remote communication minimization criteria, and compare simulation results obtained by executing our algorithm on the examples presented with the optimum cost solutions given by the ILP.

Related to our work a fair number of approaches aim at improving dependability and adaptability through influencing the software architecture. QoS-aware metadata is utilized together with Service Level Agreements (SLAs) in the planning-based middleware in the MUSIC project [5]. SLAs are common means to target policy based research allocation, e.g. [6]. The SmartFrog deployment and management framework from HP Labs describes services as collections of components and applies a distributed engine comprised of daemons running on every node in a network [7]. Fuzzy learning is applied for configuration management in server environments targeting efficient resource utilization by Xu et al. in [8]. Biologically-inspired resource allocation algorithms in service distribution problems have been targeted earlier too, such as by the authors of [9]. A different approach, namely layered queuing networks are employed by Jung et al. in an off-line framework for generating optimal configurations and policies [10]. Changing the deployment mapping of applications is investigated by others as well, however due to the fact that complexity of exact solution algorithms becomes NP-hard already in case of 2-3 hosts or several QoS dimensions applicability of these methods is restricted. Heuristics, such as greedy algorithms and genetic programming are used by Malek to maximize utility of a service from the users' perspective in [11], whereas we formulate the problem from the providers' view, while still considering user requirements. The various middleware approaches can be very good candidates serving our approach as a means of instrument for deployment that is guided by our logic.

The remainder of this paper is organized as follows. First, in Sect. 2 we discuss the component deployment problem in more detail. Next, an introduction to CEAS follows in Sect. 3. In Sect. 4 an ILP is formulated for the general component deployment problem discussed in this paper. Sect. 5 presents service examples and the corresponding simulation results are evaluated. Finally, in Sect. 6 we conclude and touch upon future work.

2 The Component Deployment Problem

We define the component deployment problem as an optimization problem, where a number $|\mathbf{C}|$ of components (labelled c_i; $i = 1, \ldots, |\mathbf{C}|$) have to be mapped to a set $\mathbf{N}$ of nodes. Components can communicate via a set $\mathbf{K}$ of collaborations (k_j; $j = 1, \ldots, |\mathbf{K}|$). We consider three types of requirements in the deployment problem. Components have execution costs e_i; $i = 1, \ldots, |\mathbf{C}|$, collaborations have communication costs f_j; $j = 1, \ldots, |\mathbf{K}|$ and some of the components can be restricted in deployment mapping. Components that are restricted to be deployed to specific nodes are called *bound* compo-

nents. Accordingly, we distinguish between the three concepts of component *mappings*, meaning a set variable obtained and refreshed in every iteration of our algorithm; component *bindings*, which represent a requirement that fixes the mapping of components to a constant value; and component *deployment* that is the actual physical placement of components to nodes, including the bound components. Physical deployment of components is triggered after the mappings obtained and refreshed in every iteration by the deployment algorithm converges to a solution satisfying the requirements.

Furthermore, we consider identical nodes that are interconnected in a full-mesh and are capable of hosting components with unlimited processing demand. We observe the processing load components hosted at a node impose and target load-balancing between the nodes available in the network. By balancing the load we mean minimizing the deviation from the global average per node execution cost. Total offered execution load is obtained from the service specification by the sum $\sum_{i=1}^{|\mathbf{C}|} e_i$, thus the global average execution cost can be obtained as

$$T = \frac{\sum_{i=1}^{|\mathbf{C}|} e_i}{|\mathbf{N}|}. \tag{1}$$

Communication costs are considered only if a collaboration between two components happens remotely, i.e. it happens between two nodes. In other words, if two components are *colocated* (are placed onto the same node) we do consider communication between them to be free (not consuming any network bandwidth).

Our deployment logic is launched with the service model enriched with the requirements specifying the search criteria and with a resource profile of the hosting environment specifying the search space. In our view, however, the logic we develop is capable of catering for any other types of non-functional requirements too, as long as a suitable cost function can be provided for the specific QoS dimension at hand. In this paper, costs in the model are constant, independent of the utilization of underlying hardware. This limitation has been removed and explored in [12] by the authors. Furthermore, we benefit from using collaborations as design elements as they incorporate local behavior of all participants and all interactions between them. That is, a single cost value can describe communication between component instances, without having to care about the number of messages sent, individual message sizes, etc. For more information on how we use collaborations for system modelling we refer to [4] and [12].

We, then define the objective of the deployment logic as obtaining an efficient (low-cost if possible optimum) mapping of components to nodes, $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$, one that satisfies the requirements in reasonable time. More importantly, the actual placement of components does not change with every iteration of the algorithm but is changed only on a larger timescale, once a mapping is converged to a solid solution to avoid churn. Re-deployment, for adaptation to changes in the execution context involves migrating components, which naturally incurs additional costs. In principle migration costs can be considered as thresholds prohibiting changing the deployment mapping if the benefit is not high enough. In this work, however, we do not consider adaptation and migration costs.

The heuristics we use in our deployment logic is guided by a cost function, $F(\mathbf{M})$ that is used to evaluate the suggested mappings iteration-by-iteration. The construction of the cost function is in accordance with the requirements of the service. How we build the corresponding cost function is discussed in Sect. 3.2.

3 Component Deployment using the Cross Entropy Ant System

3.1 The Cross Entropy Ant System

The key idea in the CEAS is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between various ant-based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet [13] uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [14]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}$ according to the cost of the solutions found. The objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$ for iteration r and $r-1$ respectively. For a tutorial on the method, [14] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies [2], such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism and self-tuned packet rate control. Additional reduction in the overhead is accomplished by pheromone sharing where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information, see [3] for details and examples on application.

In this paper, the CEAS is applied to obtain the best deployment mapping $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$ of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n \in \mathbf{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In the CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we use the deployment set $\mathbf{M}_r$ instead. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{ij,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set $\{m\}$ deployed at node n at iteration r.

In CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/|\mathbf{C}|$, where $|\mathbf{C}|$ is the number of components to be deployed, while

in the normal phase the next set is selected according to the *random proportional rule* matrix $\mathbf{p} = \{p_{mn,r}\}$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in \mathbf{M}_{n,r}} \tau_{ln,r}} \tag{2}$$

The pheromone values τ_r are updated by the backward ants as a function of the previous pheromone value τ_{r-1}, the cost of the deployment set $\mathbf{M}_r$ at iteration r, and a control variable γ_r (denoted the *temperature*) that captures the history of the cost values such that the total cost vector does not have to be stored. The CEAS function was first introduced in [1], for later extensions, details and examples see [3].

3.2 The Component Deployment Algorithm

To build a distributed cooperative algorithm, in contrast to a centralized approach, we employ the autonomous ant-like agents (denoted ants) of the CEAS method that cooperate in pursuing a common goal. Ants base their decisions solely on information available locally at a node they currently reside in. Accordingly, in our logic the information required for optimization is distributed across all participating nodes, this being a contributing factor to robustness, scalability and fault tolerance of the method.

In our deployment algorithm ants are emitted from one designated place in the network, from the so-called ant-nest. When an ant starts its random-walk over the available nodes it is assigned with the task of deploying the set of components, **C**. One round-trip (not necessarily visiting every node, but arriving back at the nest eventually) of an ant is called an iteration of the algorithm, which is repeated continuously until convergence or until the algorithm is stopped. During its random-walk the ant selects the next hop to visit entirely randomly. When arriving at a node the ant's behavior depends on if the ant is an *explorer* or a *normal* ant. The difference between the two types of ants is in the method they use to select a mapping, $m_n \subset \mathbf{M}$, for a (maybe empty) subset of **C** at each node $n \in \mathbf{N}$ they visit. Normal ants decide on whether to deploy some components at a particular node based on the pheromone database at the given node, whereas explorer ants make a decision purely randomly, thus enforcing exploration of the state-space. Explorer ants can be used both initially to cover up the search space and later, after the system is in a stable state as well to discover fluctuations in the network and, thus to aid adaptation. The useful number of initial explorer iterations is depending on the given problem size (e.g. can be estimated by the number of components in a service). Normal ants, on the other hand focus entirely on optimizing the mapping (**M**) iteration-by-iteration using and updating the pheromone tables.

An important building-block of the algorithm is the cost function applied to evaluate the deployment mapping obtained by one ant during its search. We denote the application of the cost function as $F(\mathbf{M})$. The function itself is built in accordance with the requirements. As discussed in Sect. 2, in this paper we consider load-balancing and remote cost minimization. Every ant doing its search for a mapping is capable of sampling load-levels at the nodes visited, thus the execution load imposed on the nodes in the network can be obtained by each ant as a list of sample values in the form of

$$\hat{l}_n = \sum_{c_i \in m_n} e_i, n \in \mathbf{N}, m \in \mathbf{M}. \tag{3}$$

The overall cost function evaluating the mapping obtained in an iteration using Eq. (1) and Eq. (3) then becomes

$$F(\mathbf{M}) = \sum_{n=1}^{\mathbf{N}} |\hat{l}_n - T| + \sum_{j=1}^{\mathbf{K}} I_j \; f_j \tag{4}$$

where

$$I_j = \begin{cases} 1, \text{ if } k_j \text{ remote} \\ 0, \text{ if } k_j \text{ internal to a node} \end{cases} \tag{5}$$

is an indicator function for evaluating collaborations between pairs of components.

Optimization of mappings is achieved by gradually modifying pheromone values aligned to sets of components. Pheromone values are organized into tables and are stored in every node participating in hosting the service being deployed. We use *bit-string* encoding to index the pheromone table, each entry representing a given combination/subset of components, i.e. a flag is assigned to every (unbound) component in the service model. Thus, the pheromone database has to accommodate $2^{|\mathbf{C}|}$ floating point entries using this encoding. Normalizing the entries in a node an ant can obtain a probability distribution of component sets to be mapped at the particular node. Using CEAS, which is a subclass of Ant Colony Optimization (ACO) algorithms, the optimal solution emerges in a finite number of iterations once the optimum has been observed, which does in fact happen with probability close to one in ACO systems.

Indexing of the database based on component set identifiers is illustrated in the following example. Let us start with a service provided by 4 components, i.e. $|\mathbf{C}| = 4$, $\mathbf{C} = \{c_1, c_2, c_3, c_4\}$, the database size is $2^4 = 16$. If an ant needs to collect or update information e.g. regarding the deployment of the component set $\{c_2, c_4\}$, the set labelled by the bitstring $'1010'B$ has to be addressed, which is equivalent to accessing the $'1010'B = 10^{th}$ element of the pheromone table at the node the ant resides in. The lifetime of an ant, which is equivalent to one iteration of the algorithm, contains two phases. First *forward search* is conducted during which the ant looks for a deployment mapping $\mathbf{M}$ for the set $\mathbf{C}$ visiting arbitrary nodes in the network. Once a complete mapping $\mathbf{M}$ is obtained by the ant the mapping has to be evaluated using the cost function $F(\mathbf{M})$. Using F the cost of the mapping is obtained using which the ant updates the pheromone databases in the nodes visited (the forward route has been stored in the hop-list, $\mathbf{H}$) during *forward search* during the second phase of its lifetime called *backtracking*. Once an ant is finished with backtracking and arrives back to its nest a new iteration can start and a new ant will be emitted. With convergence of the (distributed) pheromone database a strong value will emerge indicating the suggested deployment mapping, while inferior combinations will evaporate. The algorithmic steps of ants' behavior are summarized briefly in Algorithm 1.

In the algorithm we presented we have a trade-off between convergence speed and the quality of the obtained solution. However, during the deployment of services in a dynamic environment a pre-mature solution, which does satisfy the functional and non-functional requirements often suffices. In the next section we establish a centralized, off-line model to evaluate our deployment logic and to form a basis for cross-validation of the results obtained by simulation of the algorithm.

Algorithm 1 Deployment mapping of the set of components $\mathbf{C} = \{c_1, \ldots, c_{|\mathbf{C}|}\}$

1. Select next node to visit, n randomly and add n to the hop-list $\mathbf{H} = \mathbf{H} + \{n\}$.
2. Select a set of components $m_n \subseteq \mathbf{C}$ according to the random proportional rule (*normal* ant), Eq. (2), or in a random manner (*explorer* ant). If such a set cannot be found, goto step 1.
3. Update the ant's deployment mapping set, $\mathbf{M} = \mathbf{M} + \{m_n\}$.
4. Update the set of remaining components to be deployed, $\mathbf{C} = \mathbf{C} - m_n$.
5. If $\mathbf{C} \neq \emptyset$ then goto 1., otherwise evaluate $F(\mathbf{M})$ and update the pheromone values corresponding to the list of mappings $\{m_n\} \in \mathbf{M}$ going backwards along $\mathbf{H}$.
6. If stopping criteria is not met then initialize and emit new ant and goto 1.

4 An Integer Program to Validate the Algorithm

To validate the results we obtain via simulation and using various deployment scenarios we have developed an Integer Linear Program (ILP), which we solve using regular solver software. In this ILP we take into account the two counteracting objectives we presented in Sect. 2 and define a solution variable $m_{i,j}$ that will show the optimal, i.e. lowest cost mapping of components to nodes.

We start the definition of the ILP with two parameters. First $b_{i,j}$

$$b_{i,j} = \begin{cases} 1, \text{ if component } c_i \text{ is bound to node } n_j, \\ 0, \text{ otherwise.} \end{cases} \tag{6}$$

which enables the model to fix some of the mappings to cater for bound components, if any, in the model of a given service. Second, T

$$T = \lfloor \frac{\sum_{i=1}^{|\mathbf{C}|} e_i}{|\mathbf{N}|} \rfloor \tag{7}$$

that is used to approximate the ideal load-balance among the available nodes in the network. Beside the binary solution variable showing the resulting mapping, $m_{i,j}$

$$m_{i,j} = \begin{cases} 1, \text{ if component } c_i \text{ is mapped to node } n_j, \\ 0, \text{ otherwise.} \end{cases} \tag{8}$$

we utilize two additional variables. One variable for checking whether two components that communicate via a collaboration, k_i, are colocated or not, in variable col_i.

$$col_i = \begin{cases} 0, \text{ if } c_l, c_k \in k_i \text{ and } c_l \text{ is colocated with } c_k, \\ 1, \text{ otherwise.} \end{cases} \tag{9}$$

Moreover, another variable, $Delta_j$, for calculating the deviation from the ideal load-balance among the nodes participating in hosting the components.

$$\Delta_j \geq 0, \forall n_j \in \mathbf{N} \tag{10}$$

The objective function we use in the ILP is naturally very similar to Eq. (4), however to keep the model within the linearity requirement of the ILP here we use addition.

$$min \sum_{j=1}^{|\mathbf{N}|} \Delta_j + \sum_{i=1}^{|\mathbf{K}|} f_i \cdot col_i \tag{11}$$

Having established the objective function we have to define the constraints the solutions are subjected to to obtain feasible mappings. First we stipulate that there has to be one and only one mapping for all of the components.

$$\sum_{j=1}^{|\mathbf{N}|} m_{i,j} = 1, \forall c_i \in \mathbf{C} \tag{12}$$

In addition, the ILP has to take into account that some component mappings might be restricted in the case of components explicitly bound in the model. Thus, we restrict the mapping variable $m_{i,j}$ using $b_{i,j}$.

$$m_{i,j} \geq b_{i,j}, \forall c_i \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{13}$$

Next we introduce two constraints to implicitly define the values of the variable Δ_j that we apply in the objective function. We use two constraints instead of a single one to avoid having to use absolute vales (i.e. the $abs()$ function) and thus we avoid non-linear constraints.

$$\sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} - T \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{14}$$

$$T - \sum_{i=1}^{|\mathbf{C}|} e_i \cdot m_{i,j} \leq \Delta_j, \forall n_j \in \mathbf{N} \tag{15}$$

Lastly, we introduce constraints, again for implicitly building the binary variable indicating colocation of components.

$$m_{i,j} + m_{k,j} \leq (2 - col_l), k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_j \in \mathbf{N} \tag{16}$$

$$m_{i,j1} + m_{k,j2} \leq 1 + col_l, k_l = (c_i, c_k) \in \mathbf{K}, \forall c_i, c_k \in \mathbf{C}, \forall n_{j1}, n_{j2} \in \mathbf{N} \tag{17}$$

Based on this definition the ILP can be executed by a solver program and mapping costs can be obtained by submitting the appropriate data describing any given scenario corresponding to our general definition of the deployment problem in Sect. 2. The optimum mapping of components to nodes will be obtained according to the objective Eq. (11) subject to Eq. (12) – (17).

In the next section we present the example models we used in our simulations and also evaluate the mapping costs obtained by finding a lower bound using the ILP presented in this section.

5 Example Scenarios

In this section we present the three example models we simulated and validated the corresponding results for.

The first example we consider has been investigated, solved and the solutions were compared to other authors work by Widell et al. in [17]. This example originates from heuristical clustering of modules and assignment of clusters to nodes [18], which problem is NP-hard. We translate the module clustering and assignment problem to execution costs and communication costs while the complexity remains NP-hard even if

we only deal with a single service at a time. Fig. 5(a) shows the first example set of components and the collaborations between them, it comprises $|\mathbf{C}| = 10$ components interconnected by $|\mathbf{K}| = 14$ collaborations. Out of the ten components three (shaded) are bound to one of the nodes, c_2 to n_2, c_7 to n_1 and c_9 to n_3. The execution costs of components and communication costs of collaborations are shown as UML note labels.

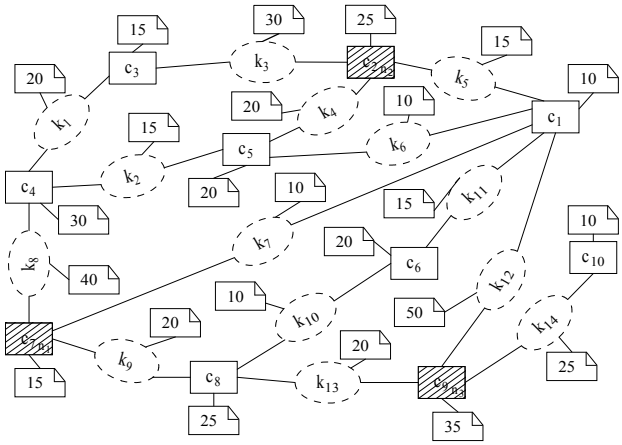


Fig. 1. Example 1, costs

The next figure (Fig. 2) shows the single optimal solution of mapping the components of the first example to three available nodes, $n_{1..3}$, i.e. it shows the optimum mapping $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{N}$. In this optimum cost mapping, the deviation from the total load-balance ($T = 68$) among the nodes are $2, -8, 7$ respectively and the mapping incurs a communication cost of 100 for the remote collaborations. Thus, this mapping results in an overall cost value equal to 117. More details are discussed about this example in [4].

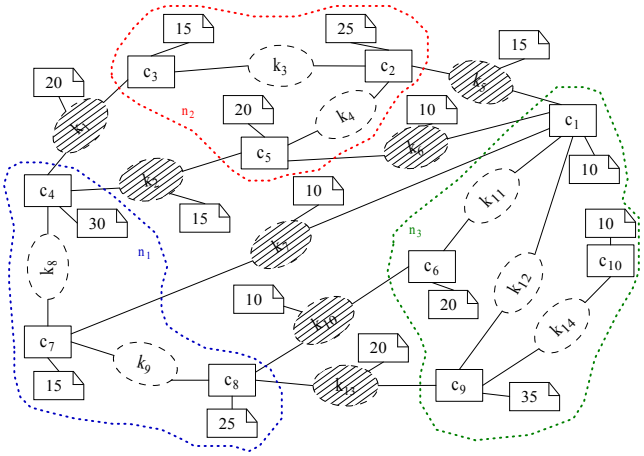


Fig. 2. Example 1, optimum mapping

The second example we considered is obtained by extending the first setting into a

larger service model and at the same time increasing the number of nodes available for deployment mapping (Fig. 3). The number of components has been increased to 15, out of which 5 components are bound, more collaborations have been introduced and one additional node is added to the deployment environment.

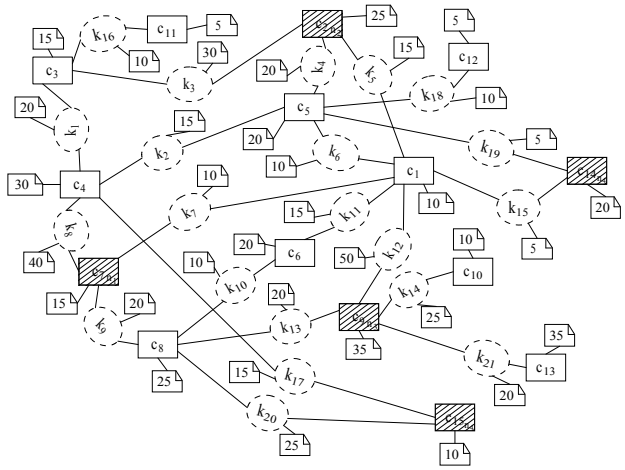


Fig. 3. Example 2, costs

This extended example is introduced to examine how our deployment algorithm behaves with the same type of problem as the computational effort needed increases or, in other words, as the solution state-space is extended.

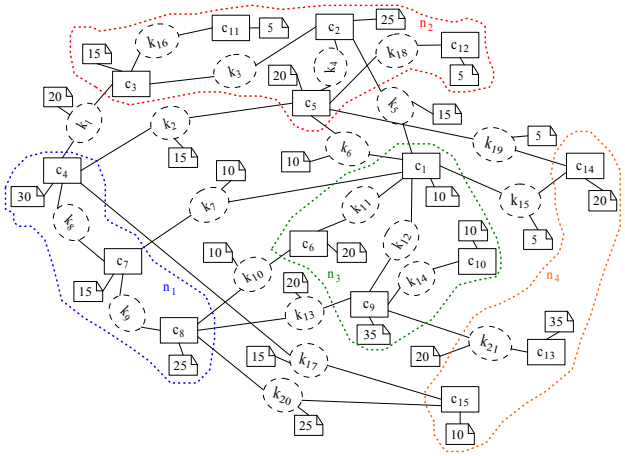


Fig. 4. Example 2, optimum mapping

The next figure, Fig. 4 presents the resulting mapping of components of the second service model to four equivalent nodes after a solution with optimum cost has been found.

In the third example the cardinality of **C** remains the same but the configuration is changed as well as the number of collaborations. In addition, the number of available nodes is increased to 6, as shown in Table 1. Due to the even more complex collaboration pattern between the components in $Example\ 3$ we omit the figure showing the

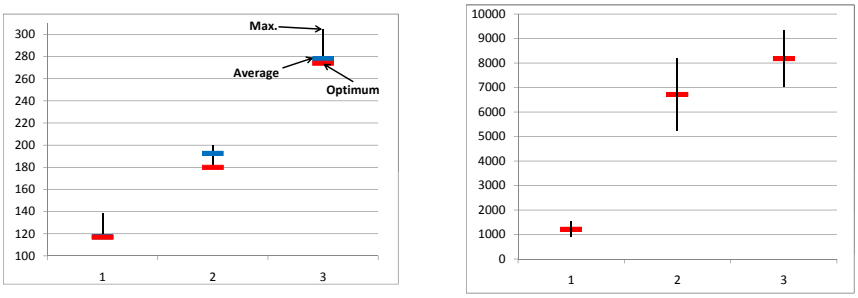
corresponding model and its optimal mapping to 6 nodes.

Table 1 also shows the absolute minimum cost values (*Optimum cost*) in the three example settings obtained by the ILP presented in Sect. 4. The presented values were generated by executing the algorithm in the simulation environment 100 times for each example model. We can see from the resulting solutions that in the first example our algorithm finds the optimum in 99 simulation runs. The somewhat larger scenario, *Example* 2 is more difficult to solve for the algorithm, thus we have a larger deviation in the mapping costs. The average cost found is slightly above the optimum as well in this case. However, it is to be noted that the adjective *slightly* is appropriate here as by changing the placement of a single component from the optimum configuration to a near-optimal one increases the costs not only by 1 but significantly more, this is also the reason for the increased deviation in this case. (In fact, the algorithm has generally found a variety of three different configurations, the optimum with cost 180 and two sub-optimal configurations with costs 195 and 200, this gave the average of 193 shown in Table 1.

Table 1. Example scenarios

	$\|\mathbf{C}\|$ (bound)	$\|\mathbf{K}\|$	$\|\mathbf{N}\|$	Optimum cost	Sim. avg.	Sim. stdev.
Example 1	10(3)	14	3	117	117.21	2.1
Example 2	15(5)	21	4	180	193	9.39
Example 3	15(5)	28	6	274	277.9	5.981

In *Example* 3 the algorithm managed to obtain solutions with costs closer to the absolute optimum obtained by the ILP, with less deviation at the same time. The main cause of this lies in the fact that the collaboration costs in the example are more fine grained, thus, the algorithm managed to find near-optimum solutions with only slightly higher costs. The average mapping costs obtained in the 100 runs and the number of *normal-ant* iterations our CEAS-based algorithm had to perform are shown in Fig. 5.



(a) Mapping costs (b) Number of iterations

Fig. 5. Simulation results for the three examples

The most significant difference for the algorithm in complexity between the original example and the extended ones lies in the size of the pheromone database needed. As the algorithm uses a binary pheromone encoding, in the first example the number of pheromone entries required is 2^7 as the number of components free to map is 7. In the larger examples, however, the pheromone database size increases to 2^{10} in each node, which results in a theoretical state-space of $4 \cdot 2^{10}$ and $6 \cdot 2^{10}$ for CEAS. Correspond-

ingly, in the experiment we applied 2000 *explorer-ants* for the initial example, but 5000 of them in the larger examples. One iteration of a centralized global-knowledge logic, such as the ILP in Sect. 4, is not really comparable with one iteration in the distributed CEAS, which is a tour made by the ant. Nevertheless, the iterations and cuts required by the ILP while solving the three example settings are shown in Table 2. The number of required (explorer and normal) iterations in CEAS is naturally higher than what is required for the ILP with a global overview. However, we advocate that we gain more by the possibility of a completely distributed execution of our algorithm and also because of the capability of adaptation to changes in the context, once the pheromone database is built up after the initial phase. For more on the adaptation capabilities of CEAS in component deployment problems we refer to [12] and [19].

Table 2. Computational effort needed by the ILP

	Example 1	Example 2	Example 3
Simplex iterations	86	495	1075
Branch and cut nodes	0	5	33

6 Closing Remarks

We presented how the deployment of distributed collaborating software components can be supported by swarm intelligence and we have introduced our ACO-based algorithm for obtaining optimal mappings of components to execution hosts. The software components we consider are described by a model enriched with relevant costs, in this particular paper with execution and communication costs. The logic we have developed can be executed in a fully distributed manner, thus, it is free of the deficiencies most existing centralized approaches suffer from, such as performance bottlenecks or single points of failure. We have showed that our algorithm is able to obtain load-balancing among the execution hosts and minimize remote communication at the same time that constitute two contradictory objectives in the deployment mapping problem.

The two main contributions of this paper are the ILP model applicable for component deployment scenarios and the cross-validation of the results obtained by our algorithm and the ILP using the three example scenarios presented. We have used the ILP to find the optimum mappings and a lower bound for the mapping costs obtained by heuristics. It is difficult, however, to compare execution times or required iterations of the two different approaches. During service deployment in a dynamic environment we are often satisfied with pre-mature solutions as long as they adhere to all the functional and non-functional requirements. ACO-based systems are proven to be able to find the optimum at least once with a probability near one, afterwards convergence to this solution is assured within a finite number of steps. CEAS, the main cornerstone in our algorithm can be considered as a subclass of ACO algorithms.

As future work we can identify several directions, out of which we are currently focusing on extending the ILP definition to cater for requirements other than the ones discussed in this paper, such as dependability aspects. Besides, we will continue validating the behavior of our deployment algorithm with extended service models and network scenarios.

References

1. B. E. Helvik and O. Wittner. Using the Cross Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks. Proceedings of 3rd International Workshop on Mobile Agents for Telecommunication Applications, 2001.
2. P. E. Heegaard and B. E. Helvik and O. J. Wittner. The Cross Entropy Ant System for Network Path Management. Telektronikk, 104(01), pp. 19-40, 2008.
3. P. E. Heegaard and O. J. Wittner. Overhead Reduction in Distributed Path Management System. Computer Networks. In Press, Accepted Manuscript, Available online 13 August 2009. Elsevier.
4. M. J. Csorba and P. E. Heegaard and P. Herrmann. Cost Efficient Deployment of Collaborating Components. In Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS). LNCS 5053, Jun 2008.
5. R. Rouvoy, et al. Composing components and services using a planning-based adaptation middleware. In Proc. of the 7th Int'l Symp. on Software Composition (SC 2008), LNCS 4954, Budapest, March 2008.
6. D. Ardagna and M. Trubian and L. Zhang. SLA based resource allocation policies in autonomic environments. Journal of Parallel and Distributed Computing 67 (2007).
7. R. Sabharwal. Grid Infrastructure Deployment using SmartFrog Technology. In Proc. of the Int'l Conf. on Networking and Services (ICNS), Santa Clara, USA, July 2006.
8. J. Xu, et al. On the use of fuzzy modeling in virtualized data center management. In Proc. of the Int'l. Conf. on Autonomic Computing (ICAC), Jacksonville, FL, USA, June 2007.
9. T. Heimfarth and P. Janacik. Ant based heuristic for OS service distribution on adhoc networks. Biologically Inspired Cooperative Computing (2006).
10. G. Jung, et al. Generating adaptation policies for multi-tier applications in consolidated server environments. In Proc. of the 5th Int'l. Conf. on Autonomic Computing (ICAC), Chicago, IL, USA, June 2008.
11. S. Malek. A User-Centric Framework for Improving a Distributed Software System's Deployment Architecture. In Proc. of the doctoral track at the 14th ACM SIGSOFT Symposium on Foundation of Software Engineering, Portland, USA, 2006.
12. M. J. Csorba and P. E. Heegaard and P. Herrmann. Adaptable model-based component deployment guided by artificial ants. In Proc. 2nd Int'l Conf. on Autonomic Computing and Communication Systems (Autonomics), ICST/ACM, Turin, September 2008.
13. G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, vol. 9, 1998.
14. R. Y. Rubinstein. The Cross-Entropy Method for Combinatorial and Continuous Optimization. Methodology and Computing in Applied Probability, 1999.
15. B. E. Helvik and O. Wittner. Using the Cross Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks. In Proc. of the 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications, 2001.
16. O. Wittner. Emergent Behavior Based Implements for Distributed Network Management. PhD thesis, Norwegian University of Science and Technology, NTNU, Department of Telematics, 2003.
17. N. Widell and C. Nyberg. Cross Entropy based Module Allocation for Distributed Systems. In Proc. of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, 2004.
18. K. Efe. Heuristic models of task assignment scheduling in distributed systems. Computer, June, 1982.
19. M. J. Csorba, H. Meling, P. E. Heegaard and P. Herrmann. Foraging for Better Deployment of Replicated Service Components. In 9th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS). LNCS 5523, Jun 2009.