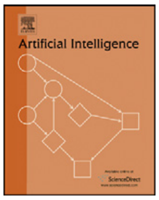
Artificial Intelligence 173 (2009) 437–465

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Computing the fault tolerance of multi-agent deployment ☆

Yingqian Zhang [a,*], Efrat Manisterski [b], Sarit Kraus [b,c], V.S. Subrahmanian [c], David Peleg [d]

[a] *Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands*
[b] *Department of Computer Science, Bar-Ilan University, Ramat Gan, 52900 Israel*
[c] *Department of Computer Science & UMIACS, University of Maryland, College Park, MD 20742, USA*
[d] *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel*

ARTICLE INFO

Article history:
Received 11 June 2007
Received in revised form 12 November 2008
Accepted 14 November 2008
Available online 25 November 2008

Keywords:
Multi-agent deployment
Fault tolerance
Algorithms
Replication

ABSTRACT

A deployment of a multi-agent system on a network refers to the placement of one or more copies of each agent on network hosts, in such a manner that the memory constraints of each node are satisfied. Finding the deployment that is most likely to tolerate faults (i.e. have at least one copy of each agent functioning and in communication with other agents) is a challenge. In this paper, we address the problem of finding the probability of survival of a deployment (i.e. the probability that a deployment will tolerate faults), under the assumption that node failures are independent. We show that the problem of computing the survival probability of a deployment is at least NP-hard. Moreover, it is hard to approximate. We produce two algorithms to accurately compute the probability of survival of a deployment—these algorithms are expectedly exponential. We also produce five heuristic algorithms to estimate survival probabilities—these algorithms work in acceptable time frames. We report on a detailed set of experiments to determine the conditions under which some of these algorithms perform better than the others.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

There have been tremendous advances in the last decade in the theory and implementation of massive multi-agent systems. However, one major obstacle to the wider deployment of multi-agent systems (MASs) is their capability of tolerating failures. MASs that are deployed across a network can quickly "go down" due to external factors such as power failures, network outages, malicious attacks, and other system issues. Protection against such unexpected failures that disable a node is critical if agents are to be used as the backbone for real world applications.

Clearly, ensuring that MASs are safe and protected involves a vast range of technologies that must authenticate users and agents, ensure secure communications, identify vulnerabilities, and identify and quarantine attacks. Our goal in this paper is far more modest, and concerns the way *replication* can form the basis of one tool (amongst many that are needed) to prevent a MAS from succumbing to failure. By replicating agents, we hope to improve the *fault tolerance* of a multi-agent system. The faults considered in this paper are those that cause disconnection (or crash) of the nodes in the network where

☆ This article is the extended version of the paper which appeared in the Second IEEE Symposium on Multi-Agent Security and Survivability [Y. Zhang, E. Manister, S. Kraus, V.S. Subrahmanian, Approximation results for probabilistic survivability, in: Second IEEE Symposium on Multi-Agent Security and Survivability, Philadelphia, USA, 2005, pp. 1–10]. This research was supported in part by the Technology Foundation STW, applied science division of NWO, and the Ministry of Economic Affairs of the Netherlands, by grant N6133906C0149, in part by ARO grant DAAD190310202, AFOSR grants FA95500610405, FA95500510298, NSF grant 0540216, NSF grant 0705587, and ISF grant 1685/07.

* Corresponding author.

E-mail addresses: yingqian.zhang@tudelft.nl (Y. Zhang), manister@macs.biu.ac.il (E. Manisterski), sarit@macs.biu.ac.il (S. Kraus), vs@cs.umd.edu (V.S. Subrahmanian), david.peleg@weizmann.ac.il (D. Peleg).

0004-3702/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2008.11.007

the MAS application resides. The fault model that we consider is one where the failure of each node in the network is represented by a probability. Given such a fault model, agents that locate on the nodes have different probabilities to be unavailable, and therefore the multi-agent system as a whole has some probability of being out of function. The idea of using replication as a fault tolerance method in our work is thus that, when facing failures, at least *one copy* of each agent will continue to reside on a connected, working host computer (node), so that the MAS as a whole can function as a unified application. Furthermore, in this paper, we focus on the problem of measuring the *probability* that a multi-agent system will tolerate the node failure. We call this probability the *survivability*[1] of a MAS system.

For example, consider the CoAX [2,35] Coalition Agent Experiment in which a large, multinational team[2] of universities, companies and government labs pieced together an experimental multi-agent application in which a set of sensory agents deployed in an ocean tracked enemy submarines. These sensory agents fed data to prediction agents that predicted when, where, and with what probability the submarine would be in a given location. Thereafter a whole set of decision making and visualization agents assisted a decision maker in determining how best to proceed. All these agents were supported, in turn, by other agents such as agents assessing trustworthiness of a source, database agents, resource discovery agents, and the like. In applications such as CoAX, it is quite likely that some nodes will “go down” or “get disconnected” from the network. Any enemy sophisticated enough to use jamming technology would also make efforts to jam the network, effectively causing some agents to have no connectivity. Thus, critical agents such as the prediction agents and the decision making agents need to be appropriately located and replicated so that the whole multi-agent system has a high probability of functioning. Of course, it is assumed that the physical hardware (sensors) are already replicated to support sensor failures—this paper does not address how to replicate physical devices.

Likewise, consider the exhaustive set of deployed multi-agent applications listed in [36]. According to their description, *Skoda*—a branch of Volkswagen—deployed an agent based production planning tool for manufacturing car engines. Their multi-agent solution looked both at low level planning and high level planning. High levels plans are examined by a set of low level planning agents that try to achieve a part of the high level plan and flag conflicts and inconsistencies in the high level plan. A back and forth process ensures, once a consensus is achieved, the production plans are sent to higher level agents who use resource allocation mechanisms to execute these plans on the production line. It is clear that in critical applications such as these, any node “going down” (for whatever reason) has the potential to cause the production line to come to a grinding halt, leading to a loss of revenue for the company.

Tichy et al. [42] describe a multi-agent system for the control of several components of a ship so as to reduce manpower requirements, while still ensuring highly reliable and survivable operation of the ship. They develop a hierarchical multi-agent architecture in which agents are embedded within hardware controllers and higher level agents coordinate and monitor the activities of groups of agent-enhanced hardware controllers. The agents are continuously engaged within a plan creation, plan commitment, and plan execution cycle. Here too, it is clear that when agents are in control of a physical environment (the ship in this case) there is high potential that the overall environment being controlled by the MAS can be adversely affected whenever an agent “goes down”. In any situation where hardware components exist (and certainly on ships), there is a possibility that hardware components will fail—for simple reasons or for more complex reasons such as the actual physical movement of the ship and/or the oceanographic and climactic conditions with which the ship is forced to contend. Thus, mechanisms are needed so that MASs can be deployed in a survivable manner even if some agents go down. Furthermore, it is important to calculate the guaranteed probability that the system will survive.

Fault tolerance and replication techniques have been extensively studied in distributed computing systems [4,10,20,31,43], but much less so in the multi-agent systems domain [5,16,29,33]. Building a fault tolerant distributed system is notoriously hard. The autonomy of agents in multi-agent systems such as CoAX makes this task even more difficult. In this paper, we build upon the framework of [27], which defined the probability that a given deployment of a MAS[3] will survive, considered the basic problem of *deployment survivability*, and proposed methods for finding most survival deployments. Zhang et al. [45] also consider the complexity of the problem of finding the *most survivable deployment*. That is, the complexity of finding the deployment with the highest survival probability, given a MAS deployment.

The model of [27] assumes *ignorance* about the dependencies between node failures. However, this assumption is not always valid. For example, an attack on Cornell’s web site is—in all likelihood—independent of the Israeli Defence Ministry’s web site going down. The framework proposed in [27] cannot handle this. The algorithm developed in [27] for finding the most survivable deployment of agents on the network only works under the ignorance assumption and has two components. The first component is an algorithm for solving the deployment survivability problem, i.e., computing the survival probability of a given deployment of the MAS, while the second component uses the first component to find the most survivable deployment. The algorithm provided in [27] for the first problem is exponential, while the second is doubly exponential. In this paper, we only focus on the first problem, of deployment survivability; the algorithm for the second component in [27] can be used directly. Moreover, instead of studying the problem under the assumption of ignorance, we assume independence between node failures.

[1] A more formal definition of survivability is given in Definition 4. See Section 6.2 for a discussion on different notations of the survivability.

[2] The team included companies such as Lockheed Martin, BBN, Qinetiq, as well as universities such as Univ. of Maryland, Univ. of Texas, Univ. of Edinburgh, and many others.

[3] We will define deployment formally later in Definition 1. For now, a deployment is simply a function that associates with each node in the network, a set of agents to be placed on that node.

The following contributions are included in this paper:

(1) In Section 2, we provide an *abstract formal model* to study the survival probability of a given deployment *under the assumption of independence of node failures.* We further show that even assuming independence, the deployment survivability problem is at least NP-hard. Moreover, it is also hard to approximate up to a factor of $2^{|V|^{1-\epsilon}}$ (where $|V|$ is the number of the nodes in the network).
(2) We show that the complexity of finding the most survivable deployment problem, even assuming independence, is at least NP-hard. We also show that for any polynomial approximation to find a sub-optimal deployment, there will be instances in which the survival probability of the most survival deployment is 1 but the algorithm returns a deployment with a survival probability of 0. Thus, any polynomial approximation algorithm is guaranteed to find at least one terrible solution.
(3) In Section 3, we introduce two centralized algorithms to accurately compute the probability that a given deployment will survive. Both algorithms take exponential time.
(4) In Section 4, we develop five different approximation algorithms to compute the probability of survival of a given deployment.
(5) About half the paper (Section 5) is devoted to a detailed comparison of the performance of the different algorithms proposed in this paper. These experiments try to identify the conditions under which one algorithm is preferable to another so that MAS applications have some foundation upon which to base a decision about which algorithm to use.
(6) Section 6 compares our algorithms with related work. We conclude by discussing the main strengths and weaknesses of the paper in Section 7.

This paper is related to three prior papers of the authors. Kraus et al. [27], as mentioned above, develop the basic MAS survivability upon which this paper is based, but do so in a setting where the relationship between failure of nodes is completely unknown. It provides a centralized algorithm to find an optimal deployment (i.e. one that maximizes the probability of survival). Our subsequent paper [41] provides a protocol by which multiple agents can dynamically re-deploy across a network when information is received that one or more nodes have gone down. Both these papers provide algorithms to actually find optimal and, when complexity does not allow so, suboptimal deployment (statically and in a centralized manner in the first case, dynamically and in a distributed manner in the second). However, when finding an optimal deployment, a common problem in both cases is to find the probability of survival of a deployment. This is a precursor to finding the deployment with the highest survival probability. It is also necessary when MAS is applied in mission critical domains. The computed survival probability can be used, for example, in deciding on whether the system is safe enough, whether to add resources and whether it is necessary to look for a new deployment (in a dynamic environment). Zhang et al. [45] takes a first step at addressing this problem. The problem presented in our work is a far more detailed and extended version of the problem described in [45].

2. An abstract probabilistic fault tolerance framework

2.1. Survivability functions

Consider a multi-agent system $\mathcal{M}$ consisting of a finite set of agents providing one or more services. We are not concerned with the framework in which the agents in $\mathcal{M}$ are encoded. For example, they could be implemented within the BDI framework or within the IMPACT [40] or some other framework or in a mix of frameworks as was the case of CoAX [2,35]. We also make no assumptions about the services provided by these agents, or the communication language they provide. We assume that $\mathcal{M}$ is deployed over a *fully* connected *overlay* network[4] $\mathcal{N} = (V, V \times V)$, where V is the set of nodes in the network. Since $\mathcal{N}$ is a fully connected network, for simplicity we denote it by its sets of vertices V. Each node $n \in V$ has some fixed amount of resources, denoted $space(n)$, that it makes available to hosting agents in a given multi-agent system. Let $space(a)$ denote the resource requirements of an agent a, and let $space(\mathcal{M}) = \sum_{a\in\mathcal{M}} space(a)$. We define a *deployment w.r.t.* $\mathcal{M}, V$ as follows.

Definition 1 *(Deployment μ).* A deployment μ w.r.t. $\mathcal{M}, V$ is a mapping from V to $2^{\mathcal{M}}$ such that $\mu(n)$ is the set of agents in $\mathcal{M}$ that are deployed at node n in the network. The deployment μ must satisfy the resource constraint, namely, $\sum_{a\in\mu(n)} space(a) \leqslant space(n)$ for each $n \in V$.

We say that μ is a *valid deployment* w.r.t. $\mathcal{M}, V$ if for each $a \in \mathcal{M}$ there is a node $n \in V$ such that $a \in \mu(n)$.

Throughout the rest of this paper, we assume that $\mathcal{M}$ is an arbitrary but fixed MAS and that V is an arbitrary but fixed set of nodes. As a consequence, we simply say "deployment μ" instead of "deployment μ w.r.t. $\mathcal{M}, V$".

[4] This is a reasonable assumption as we do not require full connectivity of the underlying physical network (merely all nodes in the physical network need to be reachable—perhaps through multiple physical links—from all other nodes).

Example 1. Consider a fully connected network with $V = \{n_1, n_2, n_3\}$ where $space(n_1) = 6$ and $space(n_2) = space(n_3) = 3$, and a MAS $\mathcal{M} = \{a_1, a_2, a_3\}$ where $space(a_i) = 4 - i$ for $i = 1, 2, 3$. A possible valid deployment μ is given by $\mu(n_1) = \{a_1, a_2, a_3\}$, $\mu(n_2) = \{a_1\}$, $\mu(n_3) = \{a_2, a_3\}$.

In a network V, any node of V can "go down" or somehow get "disconnected" from the network. In this paper, we do not go into details on for example causes of failures, failure detection method or failure rate.[5] Instead, we adopt an abstract representation on the fault model that the multi-agent system will tolerant, i.e., we assume each node has some probability of being unavailable (or out of function), due to its disconnection (or crash) from the network. We define the probability of being unavailable of a node by a *disconnect probability function*.

Definition 2 *(disconnect probability function dp).* A *disconnect probability function* (dp for short) is a mapping of $dp : V \to [0, 1]$ that assigns the probability of being disconnected to each node $n \in V$.

In the context of the CoAX application [2,35], the disconnect probability function specifies the probability that a node will somehow fall off the network—such a disconnect probability function might assign a high disconnect probability to sensory nodes deployed in the ocean, a lower disconnect probability to nodes in more secure locations in the area, and an even lower disconnect probability to highly secure nodes back in the US or Europe. In the case of Tichy's [42] application, the disconnect probabilities of agents embedded in a controller might be higher than higher level agents because of more frequent failures of hardware components. The same may be the case with the Skoda application [36].

It is important to note that the failure of a node can be permanent or temporary. Moreover, its disconnect probability $dp(n)$ can be defined in terms of time. For example, we may have a family of auxiliary functions $dp^t(n)$ that assesses the probability that node n will get disconnected in exactly t units of time. One may then derive $dp(n)$ in many ways from these $dp^t(n)$ functions. For instance, $dp(n)$ might be assumed to be the average of $dp^t(n)$'s for t ranging from 0 to some fixed upper bound.

It is also important to note that disconnect probabilities can be computed in many standard ways that are used in networking. One way is through the use of *round trip times* (RTTs) used frequently in networking. RTTs describe the time required for a packet to go from one node to another. An RTT graph looks exactly like the network itself except that each edge is labeled with the round trip time between the nodes in question. $RTT(0)$ can be initialized in any number of ways (e.g. by setting all nodes to have some fixed probability of disconnect, or by assigning such probabilities based on some a priori knowledge of the network). If $RTT(t)$ depicts the RTT graph for a given network at time t, we can compute disconnect probabilities by identifying how $RTT(t)$ differ from $RTT(t-1), RTT(t-2), \ldots,$ and so on. If a large proportion of edges associated with a node n have an increased RTT in $RTT(t)$ as compared to $RTT(t-1)$, then the disconnect probability of node n increases.

A future failure event F may cause the disconnection of a set of nodes V_F. Such a failure event will give rise to the partial network V', where $V' = V \setminus V_F$. Clearly, given the possibility of node disconnections, a partial network V' can possibly materialize in the future for any $V' \subseteq V$ (in case of a failure event F in which the nodes $V_F = V \setminus V'$ get disconnected). The probability that the future network V' will materialize is referred to as the *occurrence probability* of network V', denoted by $p_{occur}(V, V', dp)$.[6]

Consider a failure event F, and consider the deployment μ restricted to the future network V' for $V' = V \setminus V_F$. Clearly, μ still satisfies the resource constraint. However, the deployment μ may no longer be valid w.r.t. V' (i.e., some agents in $\mathcal{M}$ may not be deployed in any node of V'). We say that the future network V' is valid if this does not happen. Formally, we have the following definition.

Definition 3 *(Valid future network).* Given a deployment μ and a network V, a possible future network V_i is *valid* if and only if μ is a *valid deployment* w.r.t. V_i, i.e., for each agent $a \in \mu$, $\{n \mid a \in \mu(n)\} \cap V_i \neq \emptyset$.

The set of possible valid future networks of a network V is defined by $ValidV(\mu) = \{V_i \mid V_i \subseteq V$ and μ is valid with respect to $V_i\}$.

We say that the system $(\mathcal{M}, V, dp, \mu)$ survives the failure event F (where the set of nodes V_F gets disconnected) if the remaining $V' = V \setminus V_F$ is a valid network.

We are ready to define an abstract notion of the survivability function as given below.

Definition 4 *(Survivability function).* Consider a fixed multi-agent system $\mathcal{M}$ and a network V. A *survivability function* $SF(\mathcal{M}, V, dp, \mu)$ maps a deployment μ and a disconnect probability function dp to the probability that the system $(\mathcal{M}, V, dp, \mu)$ will survive. When $\mathcal{M}$, V and dp are clear from the context, we may denote the survivability simply as $SF(\mu)$.

[5] See Section 6.4 for a more detailed discussion on the failure mode.
[6] Whenever V and dp are clear from the context, we denote the occurrence probability of V' simply by $p_{occur}(V')$.

The survival probability of μ is obtained by summing up the occurrence probabilities of all its possible valid future networks.

$$SF(\mathcal{M}, V, dp, \mu) = \sum_{V_i \in ValidV(\mu)} p_{occur}(V, V_i, dp). \tag{1}$$

We call the survival probability of a given deployment μ, $SF(\mu)$, the *survivability* of deployment μ.

Kraus et al. [27] use a linear programming model to define the survival probability of a MAS under the *ignorance assumption*, i.e., assuming we are completely ignorant about node failure dependencies. However, this assumption may not be valid for many multi-agent applications, where the hosts are geographically distributed, as there can be simple or complex dependencies (or independence) between failures of different hosts. For example, the failure of a node in Australia is likely to be independent of the failure of a node in Maryland, in which case the independence assumption may be more appropriate than the ignorance assumption. In the case of the CoAX application, for example, if a node n goes down, the probability that a node n' will go down depends on various factors. For instance, if n, n' are both in the area of the underwater sensor array, then the failure of n is likely to be positively correlated with the failure of n'. However, if n is in the region of the underwater sensor array, and n' is in the UK, the failures of these nodes will probably be independent. It is therefore apparent that there is a wide array of possible ways in which the failure of a node is related to the failure of another node. Likewise, in the case of the Skoda multi-agent application, it may well be the case that the failure of agents associated with planning are independent of failures of agents associated with the monitoring and execution process as the latter are likely to directly control (or sense) physical devices, while the former do not. The same could be the case for Tichy's ship control [42] application.

Kraus et al. [27] study one extreme—where there is complete ignorance of the relationship between node failures. This ignorance assumption causes all survival probabilities to be extraordinarily pessimistic (low). In addition, they [27] show that the most survivable deployment problem (namely, finding a deployment μ^* which maximizes $SF(\mathcal{M}, V, dp, \mu)$ for given $\mathcal{M}, V, dp$) is intractable under the ignorance assumption.

Since there are many cases where the independence assumption is valid (as in the example mentioned above), throughout this paper, we develop the survivability algorithms under the following independent assumption:

Assumption 1 *(Failure independence assumption).* Given a network V, node failures are *independent* of one another.

Later in the paper, we will discuss how the techniques in this paper can be extended in order to remove the independence assumption.

Example 2. Consider the network and deployment given in Example 1, and suppose the disconnect probability function dp is given by $dp(n_1) = 0.7$, $dp(n_2) = 0.6$, $dp(n_3) = 0.4$. The *possible valid future networks* are $V_1 = \{n_1\}$, $V_2 = \{n_2, n_3\}$, $V_3 = \{n_1, n_2, n_3\}$, $V_4 = \{n_1, n_2\}$, $V_5 = \{n_1, n_3\}$. The occurrence probability of V_1 is $p_{occur}(V_1) = 0.3 \cdot 0.6 \cdot 0.4 = 0.072$, and similarly, $p_{occur}(V_2) = 0.168$, $p_{occur}(V_3) = 0.072$, $p_{occur}(V_4) = 0.048$ and $p_{occur}(V_5) = 0.108$. The survivability of the deployment is given by $SF(\mathcal{M}, V, dp, \mu) = 0.072 + 0.168 + 0.072 + 0.048 + 0.108 = 0.468$.

2.2. Complexity results for survivability

In this section, we investigate the complexity of the deployment survivability problem (namely, we compute the survivability of a given MAS deployment) by replacing the ignorance assumption [27] with the failure independent Assumption 1.

Given a network V, a multi-agent application $\mathcal{M}$, a disconnect probability dp and a deployment μ, the *occurrence probability* of network V' (*under the independence assumption*) can be calculated by the following equation:

$$p_{occur}(V, V', dp) = \prod_{n_p \in V'} \big(1 - dp(n_p)\big) \cdot \prod_{n_q \in V \setminus V'} dp(n_q).$$

One might expect that the deployment survivability problem would be easier with the assumption of independence. However, the following result shows that this problem is at least NP-hard even under the independence assumption, and moreover, it is also hard to approximate up to a factor of $2^{|V|^{1-\epsilon}}$.

Theorem 5. *Computing the survivability $SF(\mathcal{M}, V, dp, \mu)$ of a given deployment μ is at least NP-hard, and it is also hard to approximate up to a factor of $2^{|V|^{1-\epsilon}}$.*

Proof. By a reduction from the problem of finding the number of satisfying truth assignments of a given monotone CNF formula φ. An instance of this problem is a formula φ in (monotone) CNF form over K Boolean variables $x_1 \ldots x_K$, namely, a conjunction of M clauses, $\varphi = C_1 \cap \cdots \cap C_M$, where each clause $C_i = (x_{i_1}, \ldots, x_{i_l})$ is a disjunction of literals, and all literals are nonnegated. Given such an instance φ, we create an instance of the deployment survivability problem as follows. For each clause C_j we create an agent a_j. For each logical variable x_i we create a node n_i. An agent a_j is deployed on all nodes corresponding to literals that occur in C_j, i.e., $a_j \in \mu(n_i)$ if x_i appears in C_j. The disconnect probability of each node $n \in V$ is set to $dp(n) = 0.5$.

It is easy to see that the survivability of μ is $Z/2^{|V|}$ if and only if there are Z truth assignments of the (monotone) CNF formula φ.

It is well-known that counting the number of satisfying truth assignments of a given monotone CNF formula is NP-hard. It is even NP-hard to approximate this number up to a factor of $2^{K^{1-\epsilon}}$, where K is the number of variables. This is true even if each clause of the formula contains two variables [38]. Consequently, the deployment survivability problem is NP-hard, and it is also hard to approximate up to a factor of $2^{|V|^{1-\epsilon}}$. □

Next we show that the most survivable deployment problem, namely, finding a deployment μ^* which maximizes $SF(\mathcal{M}, V, dp, \mu)$ for given $\mathcal{M}, V, dp$, is at least NP-hard. Consequently, it may be interesting to look for a polynomial time heuristic algorithm that is guaranteed to output a deployment with a survival probability within ϵ of the optimal deployment μ^*, for some $\epsilon > 0$. Unfortunately, the following theorem also states that the best ϵ is 1 (under the assumption that $\mathsf{P} \neq \mathsf{NP}$).

Theorem 6. (1) *Finding an optimal* (*most survival*) *deployment μ^* for a given $\mathcal{M}$, V, dp is at least NP-hard even under the independence assumption.*

(2) *If* $\mathsf{P} \neq \mathsf{NP}$, *then for every polynomial approximation to find a sub-optimal deployment there are instances in which the survival probability of μ^* is* 1 *but the algorithm returns a deployment with a survival probability of* 0.

Proof. It is suffice to prove claim (2), since claim (1) follows immediately from it. Suppose that claim (2) is false. Then a polynomial algorithm AL exists such that it always returns a deployment with a survival probability larger than 0, when the survival probability of the optimal deployment is 1. We will use AL in order to obtain a polynomial time algorithm for solving the (NP-complete) "subset sum" problem. This problem requires one to decide, given a finite set $S \subset N$ and a target integer $K \in N$, whether there exists a subset $S' \subseteq S$ whose elements sum up to K [9].

Given a set $S = \{s_1, \dots, s_n\}$ and a target K, construct the following 2-node network $\mathcal{N}_{S,K}$. Each member of the set $s \in S$ is represented by an agent a_s, whose space requirement is s, $space(a_s) = s$. The two nodes n_1 and n_2 have $space(n_1) = K$ and $space(n_2) = \sum_{s \in S} s - K$. Assume the network is reliable, i.e., the disconnect probabilities are $dp(n_i) = 0$ for $i = 1, 2$. It is easy to see that the survivability of the optimal deployment μ^* is

$$SF(\mathcal{M}, \mathcal{N}_{S,K}, dp, \mu^*) = \begin{cases} 1, & \exists \text{ a subset } S' \subseteq S \text{ s.t. } \sum_{s' \in S'} s' = K, \\ 0, & \text{otherwise.} \end{cases}$$

Therefore the following algorithm solves the subset sum problem:

- For given S and K, build the network $\mathcal{N}_{S,K}$ as described above.
- Run algorithm AL on $\mathcal{N}_{S,K}$.
- If AL returns a deployment with a survival probability greater than 0, then return Yes, else return No. □

3. Algorithms for computing exact deployment survivability

This section describes two algorithms for computing the survivability of a given deployment. Algorithm $SF1_n$ is a "naive" algorithm which is exponential in the number of nodes (and is therefore suitable for use when $|V|$ is small), while Algorithm $SF1_a$ is exponential in the number of agents (and hence is suitable when $|\mathcal{M}|$ is small).

3.1. The naive Algorithm $SF1_n$

This algorithm uses the definition of Eq. (1) directly in order to calculate the survivability of μ. More explicitly, it enumerates all possible valid future networks V_i, and computes their occurrence probabilities. Finally, it returns the survival probability of μ, namely, the sum of the occurrence probabilities of all possible valid future networks.

3.2. The agent-based Algorithm $SF1_a$

Given a deployment μ, let A_{a_i} be the event that all the nodes that agent a_i is deployed on are disconnected. Let A_d be the event that at least one of the A_{a_i} events occurs. The probability that event A_{a_i} will occur is given by $Pr(A_{a_i}) = \prod_{a_i \in \mu(n)} dp(n)$. In order for μ to survive, none of the A_{a_i} events should occur. Unfortunately, the A_{a_i} events are not mutually exclusive. Thus, in order to compute the survivability of μ using A_{a_i} we need to calculate the probability of the disjunction of non-mutually exclusive events using the inclusion–exclusion formula as presented below.

Definition 7. Suppose μ is a deployment w.r.t. an overlay network V and suppose the node disconnect probabilities are independent. Then

$$SF1_a(\mathcal{M}, V, dp, \mu) = 1 - Pr(A_d) \tag{2}$$

where

$$Pr(A_d) = Pr(A_{a_1} \vee A_{a_2} \vee \cdots \vee A_{a_{|\mathcal{M}|}})$$
$$= \sum_{a_i \in \mathcal{M}} Pr(A_{a_i}) - \sum_{a_i \neq a_j,\ a_i, a_j \in M} Pr(A_{a_i} \wedge A_{a_j}) + \cdots + (-1)^{|\mathcal{M}|+1} Pr(A_{a_1} \wedge \cdots \wedge A_{a_{|\mathcal{M}|}}). \tag{3}$$

Algorithm $SF1_a$ calculates the probabilities of all the A_{a_i} events and then computes the above formula and returns its result.

Example 3. Consider the network and deployment given in Example 1 and consider the disconnect probability function given in Example 2. Agent a_1 is located on nodes n_1 and n_2, and the probability that both will get disconnected is $Pr(A_{a_1}) = dp(n_1)dp(n_2) = 0.7 \times 0.6 = 0.42$. Similarly, $Pr(A_{a_2}) = Pr(A_{a_3}) = dp(n_1)dp(n_3) = 0.28$; $Pr(A_{a_1} \wedge A_{a_2}) = Pr(A_{a_1} \wedge A_{a_3}) = dp(n_1)dp(n_2)dp(n_3) = 0.168$; $Pr(A_{a_2} \wedge A_{a_3}) = 0.28$; $Pr(A_{a_1} \wedge A_{a_2} \wedge A_{a_3}) = 0.168$.

Thus, the survivability of the deployment is given by

$$SF1_a(\mathcal{M}, V, dp, \mu) = 1 - \big(Pr(A_{a_1}) + Pr(A_{a_2}) + Pr(A_{a_3})\big) + \big(Pr(A_{a_1} \wedge A_{a_2}) + Pr(A_{a_1} \wedge A_{a_3}) + Pr(A_{a_2} \wedge A_{a_3})\big)$$
$$- Pr(A_{a_1} \wedge A_{a_2} \wedge A_{a_3}) = 0.468.$$

Efficiency can be improved by using the idea presented in [27] to reduce the number of agents and nodes without any loss of accuracy. For this purpose, denote the nodes in which an agent a_i is located by $Loc(a_i)$. In [27], Kraus et al. prove that the survivability of a deployment is *unaffected* if we eliminate *irrelevant agents*—an agent a is irrelevant if any other agent a' exists such that it is deployed in a subset of nodes in which a is deployed, that is, $Loc(a') \subseteq Loc(a)$. Throughout this paper, when computing survivability with any algorithm, we always apply this method first to eliminate the irrelevant agents, and then carry out the computation on the simplified deployments.[7] The following example gives the reader a quick idea of how the elimination idea works.

Example 4. Consider the network, deployment and the disconnect probability function given in Example 3. We have $Loc(a_1) = \{n_1, n_2\}$, $Loc(a_2) = \{n_1, n_3\}$, and $Loc(a_3) = \{n_1, n_3\}$. As $Loc(a_3) \subseteq Loc(a_2)$, we may remove a_3 from the deployment μ and update μ as follows: $\mu'(n_1) = \{a_1, a_2\}$, $\mu'(n_2) = \{a_1\}$, and $\mu'(n_3) = \{a_2\}$. We compute the survivability of μ' by Algorithm *SF1a* as

$$SF1_a\big(\mathcal{M} \setminus \{a_3\}, V, dp, \mu'\big) = 1 - \big(Pr(A_{a_1}) + Pr(A_{a_2})\big) + Pr(A_{a_1} \wedge A_{a_2}) = 1 - (0.42 + 0.28) + 0.168 = 0.468.$$

Clearly, $SF1_a(\mathcal{M} \setminus \{a_3\}, V, dp, \mu') = SF1_a(\mathcal{M}, V, dp, \mu)$.

3.3. An upper bound of the survivability

As Algorithms $SF1_a$ and $SF1_n$ both take exponential time, we now establish an upper bound for the survivability of μ based on Algorithm $SF1_a$. This upper bound can be used to evaluate heuristics proposed later in the paper.

As mentioned above, A_d is the event that all nodes on which some agent is located get disconnected. We are therefore interested in the complement of event A_d. Finding a lower bound for $Pr(A_d)$ and subtracting it from 1 yields an upper bound on the survivability of μ. Given Eq. (3), the Bonferroni inequalities [18] state that if the sum on the right is truncated after k terms ($k < |\mathcal{M}|$), then the truncated sum is an *upper bound* of $Pr(A_d)$ if k is odd and is a *lower bound* of $Pr(A_d)$ if k is even. For example, it is easy to see that $\sum_{a_i \in \mathcal{M}} Pr(A_{a_i}) - \sum_{a_i \neq a_j, a_i, a_j \in \mathcal{M}} Pr(A_{a_i} \wedge A_{a_j})$ (where $|\mathcal{M}| \geqslant 2$) is a lower bound for $Pr(A_d)$. This lower bound can be calculated incrementally until we run out of a predefined maximal running time or the difference between what we add to the expression (an odd term) and what we subtract from the expression (an even term) is very small. We can then take the maximum of all the lower bounds that we computed. Subtracting this value from 1 will give us an upper bound on the survivability of μ. The following algorithm[8] explains the way to find an upper bound of the survivability of μ.

Algorithm 1 *(UB($\mathcal{M}, V, dp, \mu, D, TM$)).*

($*$ Input: a predefined value D and a maximum running time *TM* $*$)

[7] Another simplification of the deployments presented in this paper is that we assume the number of copies of an agent on one node is at most one. Note in our model, the survivability of a deployment, where more than one copy of the same agents exist on the same nodes, is equal to the survivability of the simplified deployment.

[8] All our algorithms receive a set of agents $\mathcal{M}$, a set of nodes V, a disconnect probability function dp and a MAS deployment μ as input. For convenience, in the input part of each algorithm we will describe only the additional input parameters.

```
(∗ Output: an upper bound on the survivability of $\mu$ ∗)

(1) start timer $T$, $k = 1$; (* $k$ specifies the number of elements in the subsets *)
(2) $value = 0$, $value_o = 1$;
(3) $p = 0$, $minub = 1$, $maxlb = 0$;
(4) while $((|value - value_o| \geqslant D)$ and $(T < MT))$
      (a) $value_o = value$;
      (b) $value = ksubset(\mu, k)$; (* returns the sum of the probability of $k$-subsets *)
      (c) if $k$ is odd, then $sign = 1$;
          else $sign = -1$;
      (d) $p = p + sign \times value$;
      (e) if $(sign = 1)$
            if $p < minub$, then $minub = p$;
      (f) else
            if $p > maxlb$, then $maxlb = p$;
      (g) $k = k + 1$;
(5) return $(1 - maxlb)$.
```

Proposition 8. *Given a deployment μ and a network V, Algorithm UB yields an upper bound on the survivability of μ.*

Proof. Since Algorithm $SF1_a(\mathcal{M}, V, dp, \mu)$ returns the exact survivability of μ, we still need to show that $UB(\mathcal{M}, V, dp, \mu, D, TM) \geqslant SF1_a(\mathcal{M}, V, dp, \mu)$. Define $S_1 = \sum_{i=1}^{|\mathcal{M}|} Pr(A_{a_i})$, $S_2 = \sum_{i<j} Pr(A_{a_i} \wedge A_{a_j})$, and $S_k = \sum_{i_1<i_2<\cdots<i_k} Pr(A_{a_{i_1}} \wedge \cdots \wedge A_{a_{i_k}})$ for $2 < k \leqslant |\mathcal{M}|$. Based on Eqs. (2), (3) and Bonferroni inequalities [18], we have $SF1_a(\mathcal{M}, V, dp, \mu) = 1 - Pr(\bigcup_{i=1}^{|\mathcal{M}|} A_{a_i}) \leqslant 1 - \sum_{j=1}^{k} (-1)^{j+1} S_j$ (for even $k \geqslant 2$) $\leqslant 1 - maxlb = UB(\mathcal{M}, V, dp, \mu, D, TM)$. □

4. Heuristic algorithms for computing survivability

As Algorithms $SF1_n$ and $SF1_a$ are too expensive for real-world applications, we now propose several heuristics to compute the survivabilities of deployments. We are interested in finding *lower bounds* for $SF(\mu)$, which will allow us to guarantee that a given deployment μ has a survival probability that exceeds some threshold.

4.1. An anytime Algorithm SF2

Algorithm $SF1_a$ can be turned into an anytime algorithm using the same idea used to compute the upper bound. If we find an upper bound for $Pr(A_d)$ and subtract it from 1, then we attain a lower bound on the survivability of μ. Again, looking at Eq. (3), $\sum_{a_i \in \mathcal{M}} Pr(A_{a_i})$ is an upper bound for $Pr(A_d)$. Any odd number of terms of Eq. (3) provides an upper bound. An anytime algorithm can iteratively add terms until we exceed a time deadline or the ratio between the maximum among the lower bounds and the minimum among the upper bounds is smaller than a specified ratio r. The algorithm is similar to Algorithm 1 except that the anytime algorithm returns $(1 - minub)$ as the lower bound on the survivability of the deployment.

Algorithm 2 *(Algorithm $SF2(\mathcal{M}, V, dp, \mu, R, TM)$).*

```
(∗ Input: a predefined ratio $R$ and a maximum running time $TM$ ∗)
(∗ Output: a lower bound on $SF(\mu)$ ∗)

(1) start timer $T$, $k = 1$; (* $k$ specifies the number of elements in the subsets *)
(2) $p = 0$, $minub = 1$, $maxlb = 0$;
(3) while $(\frac{maxlb}{minub} \geqslant R)$ and $(t < MT)$
      (a) $value = ksubset(\mu, k)$; (* returns the sum of the probability of k-subsets *)
      (b) if $k$ is odd, then $sign = 1$;
          else $sign = -1$;
      (c) $p = p + sign \times value$;
      (d) if $(sign = 1)$
            if $p < minub$, then $minub = p$;
      (e) else
            if $p > maxlb$, then $maxlb = p$;
      (f) $k = k + 1$
(4) return $(1 - minub)$.
```

Proposition 9. *Given a deployment μ and a network V, the anytime Algorithm SF2 returns a lower bound on $SF(\mu)$.*

Proof. The proof is similar to that of Proposition 8, except here we need to show that $SF2(\mathcal{M}, V, dp, \mu, R, TM) \leqslant SF1_a(\mathcal{M}, V, dp, \mu)$. Again, define $S_k = \sum_{i_1<i_2<\cdots<i_k} Pr(A_{a_{i_1}} \wedge \cdots \wedge A_{a_{i_k}})$ for $2 < k \leqslant |\mathcal{M}|$. Based on Eqs. (2), (3) and Bonferroni inequalities [18], we have $SF1_a(\mathcal{M}, V, dp, \mu) = 1 - Pr(\bigcup_{i=1}^{|\mathcal{M}|} A_{a_i}) \geqslant 1 - \sum_{j=1}^{k}(-1)^{j+1}S_j$ (for odd $k \geqslant 1$) $\geqslant 1 - minub = SF2(\mathcal{M}, V, dp, \mu, R, TM)$. □

4.2. A tree-based Algorithm SF3

Algorithm *SF*3 is a heuristic which provides a lower bound on $SF(\mu)$. Algorithm $SF1_n$ computes the survivability of μ by summing up the occurrence probabilities of all possible valid future networks V_i. The high complexity of Algorithm $SF1_n$ is therefore caused by the fact that it enumerates and checks an exponential number of possible future networks. In contrast, Algorithm *SF*3 attempts to check only a bounded number of V_i's. This immediately implies that $SF3(\mathcal{M}, V, dp, \mu) \leqslant SF(\mathcal{M}, V, dp, \mu)$, that is, Algorithm *SF*3 yields a lower bound on $SF(\mu)$. Obviously, in order to make this lower bound as close as possible to $SF(\mu)$, the selection process should try to pick the future networks V_i whose occurrence probability is as large as possible.

Algorithm *SF*3 does this via a *tree search* in which every vertex is labeled with a subset of V. The algorithm starts from the root of the tree, labels it with V and computes V's occurrence probability. For every $n \in V$, there is a vertex labeled $V \setminus \{n\}$ in the second level of the tree. For each label V' of such a vertex, the algorithm checks if V' is valid and computes its occurrence probability. However, only the α vertices with the highest occurrence probabilities on the second level of the tree are further expanded in the same way. If a vertex labeled V_i is expanded, its children will be labeled by $V_i \setminus \{n\}$ for $n \in V_i$. Again, only α vertices on each tree level will be expanded. The algorithm stops when there are no more vertices to expand, and returns the sum of the occurrence probabilities of all the valid future networks occurring as labels in the search tree.

If α is polynomial in the size of the input, then Algorithm *SF*3 considers only a polynomial number of future networks. Therefore it may return poor results if there is a large number of nodes. For example assume that the disconnect probability of nodes is distributed uniformly in $[0, 0.5]$. The output of Algorithm *SF*3 is bounded from above by the number of subsets considered multiplied by the largest occurrence probability. The largest occurrence probability in this case is bounded by $\prod_{n\in V}(1 - dp(n))$. Therefore the survivability estimate given by Algorithm *SF*3 is usually no greater than $\alpha 0.9^{|V|}$, which is smaller than $10^{-\frac{|V|}{22}}$.

Since the performance of Algorithm *SF*3 could be very poor, we propose two heuristics to improve its value.

Disjoint removal heuristic: The first heuristic is that prior to running Algorithm *SF*3, we check for each agent $a \in \mathcal{M}$ whether the nodes in $Loc(a)$ are disjoint from those in $Loc(a')$ for any other agent a'. Let $\mathcal{M}'$ denote the set of such agents a' with disjoint sets. We can compute the survivability of $\mathcal{M}'$ directly by $SF(\mathcal{M}', V, dp, \mu) = \prod_{a\in\mathcal{M}'}(1 - \prod_{a\in\mu(n), n\in V} dp(n))$. We then apply Algorithm *SF*3 on the remaining agents $\mathcal{M} \setminus \mathcal{M}'$. At the end of the algorithm, we multiply the returned value by $SF(\mathcal{M}', V, dp, \mu)$, i.e., $SF3(\mathcal{M}, V, dp, \mu) = SF(\mathcal{M}', V, dp, \mu) \cdot SF3(\mathcal{M} \setminus \mathcal{M}', V, dp, \mu)$.

Node removal heuristic: The second heuristic is based on the idea that if the number of nodes involved in Algorithm *SF*3 is larger than some predefined constant K_b (i.e. $|V| > K_b$), then we may reduce it by removing some nodes which contribute less to the survival of the μ. We sort the nodes in ascending order of $\sqrt[\rho]{1 - dp(n)}$, where ρ denotes the number of agents on node n. The first K_r nodes can be deleted from the deployment. The intuition behind this formula is to remove nodes whose disconnect probability is relatively higher, since the occurrence probabilities of networks that include these nodes are relatively low. In addition we want to remove nodes that are deployed with a relatively small number of agents, since the disconnection of these nodes influences fewer agents (compared to nodes that are deployed with a larger number of agents). Note that as $dp(n)$ increases $\sqrt[\rho]{1 - dp(n)}$ decreases. In addition as ρ increases $\sqrt[\rho]{1 - dp(n)}$ increases, since $1 - dp(n)$ is a fraction. The first K_r nodes can be deleted from the deployment. Note that after the removal action, it may be possible to eliminate more irrelevant agents by applying the idea presented in [27] (see Example 4) in order to further simplify the computation.

Algorithm *SF*3 is presented below.

Algorithm 3 *(Algorithm SF3($\mathcal{M}, V, dp, \mu, \alpha, K_b, K_r$)).*

```
(* Input: (1) the predefined number of selected vertices $\alpha$ *)
(* (2, 3) the predefined constants $K_b$ and $K_r$ *)
(* Output: a lower bound on $SF(\mu)$ *)

(1) $disjoint_{surv} = rmvdisjoint(\mu, V, \mathcal{M}, dp)$;
      (* remove the agents with disjoint locations, return the survivability of the removed agent set as described above *)
(2) if $|V| > K_b$, then
          $V = rmvnodes(\mu, V, \mathcal{M}, dp, K_r)$;
      (* remove $K_r$ nodes, and update the network as described above *)
```

```
(3) $best_{val} = \prod_{n \in V}(1 - dp(n))$;
      (* compute the occurrence probability of the network $V$ *)
(4) $temp = \{V\}$, $done = false$, $flag = false$;
(5) **while** ($\neg done$ ), **do**
      (a) $X' = \emptyset$;
      (b) **while** ($temp \neq \emptyset$)
            (i) $X = headof(temp)$, $temp = temp \setminus X$;
            (ii) $X' = X' \cup \{X \setminus \{x_i\} \mid x_i \in X\}$;
      (c) $S_{valid} = \emptyset$;
      (d) **while** ($X' \neq \emptyset$), do (* remove invalid sets and repetitive sets in $X'$ *)
            (i) $V' = headof(X')$, $X' = X' \setminus V'$;
            (ii) **if** $V' \notin S_{valid}$ and $V' \in ValidV(\mu)$, then
                  $S_{valid} = S_{valid} \cup V'$, $flag = true$;
      (e) **if** ($\neg flag$), then $done = true$;
          **else, do**
            (i) **for each** ($V_i \in S_{valid}$)
                  $p_{occur}V_i = \prod_{n \in V_i}(1 - dp(n)) \cdot \prod_{n \in V \setminus V_i} dp(n)$;
                  $best_{val} = best_{val} + p_{occur}V_i$;
            (ii) $S_{valid} = sort(S_{valid}, p_{occur}V_i)$;
                  (* sort sets in $S_{valid}$ in descending order according to their occurrence probabilities *)
            (iii) **for** ($j = 0$, $j < \alpha$, $j++$) (* keep the first $\alpha$ of sets *)
                  • $temp = temp \cup headof(S_{valid})$;
                  • $S_{valid} = S_{valid} \setminus headof(S_{valid})$;
            (iv) $flag = false$;
(6) return ($best_{val} \times disjoint_{surv}$).
```

In the algorithm, the function *rmvdisjoint*$(\mu, V, \mathcal{M}, dp)$ implements the disjoint removal heuristic described earlier which removes agents whose set of locations is disjoint from the set of locations of any other agent. The function *rmvnodes*$(\mu, V, \mathcal{M}, dp, K_r)$ implements the node removal heuristic, which reduces the number of the nodes by removing K_r nodes which contribute less to the survival of the deployment.

The following proposition states that Algorithm *SF*3 is a correct polynomial time approximation of $SF(\mu)$.

Proposition 10. (1) *For any $\alpha > 0$, the value returned by Algorithm SF3 is a lower bound for $SF(\mu)$.*

(2) *Suppose α is fixed. Then the time complexity of Algorithm SF3 is $O(\alpha|V|^2 \log(\alpha|V|) + \alpha|V|^2|\mathcal{M}|)$, i.e., the algorithm is polynomial if α is polynomial in the size of the input.*

A sketch of the proof of the above proposition is given below.

Proof. Suppose the search starts from the root in level 0 of the tree, labeled by the set of nodes V. Consider a vertex z in level i of the tree, labeled by a set V' of size $|V'| = |V| - i$. In the next level of the tree, level $i+1$, z has $|V'|$ children, each of whose labels are generated by removing exactly one node from V'. Only the α valid sets with the highest occurrence probabilities in level i are used to generate the next level. Thus the total number of vertices at level $i+1$ is $\alpha(|V| - i)$. For each generated set of nodes, the occurrence probability of a set can be computed in $O(1)$[9] time, and checking if the set is valid (i.e., if it contains all agents) can be done in $O(|\mathcal{M}|)$ time. In addition, for each level, sorting the sets of the nodes takes $O(\alpha|V|\log(\alpha|V|))$. As the maximum depth generated in the tree is $|V|$, the time complexity of Algorithm *SF*3 is:

$$O\big(|V|\big(\alpha|V||\mathcal{M}| + |V|\alpha|V|\log\big(\alpha|V|\big)\big)\big) = O\big(\alpha|V|^2\log\big(\alpha|V|\big) + \alpha|V|^2|\mathcal{M}|\big).$$

Since the survivability of μ is the sum of the occurrence probabilities of all the valid subsets while Algorithm *SF*3 only considers a subset of all valid subsets, clearly $SF3(\mathcal{M}, V, dp, \mu, \alpha, K_b, K_r) \leqslant SF(\mathcal{M}, V, dp, \mu)$ for any value of α, K_b and K_r. □

Example 5. Consider the network and the updated deployment of Example 4, where $\mu(n_1) = \{a_1, a_2\}$, $\mu(n_2) = \{a_1\}$, and $\mu(n_3) = \{a_2\}$. Suppose $\alpha = 1$, $K_b = 20$ and $K_r = 3$. The root is $V = \{n_1, n_2, n_3\}$. Thus $X_{s0} = \{n_1, n_2, n_3\}$, and $p_{occur}(X_{s0}) = (1 - dp(n_1))(1 - dp(n_2))(1 - dp(n_3)) = 0.072$.

In the next level of the graph, three subsets are generated by removing one node from V:

[9] This can be done by using the occurrence probability of the parent's vertex. This is due to the fact that the difference between a vertex in the tree labeled $V \setminus \{n\}$ and its parent's vertex in the tree labeled network V is that in $V \setminus \{n\}$, a node n doesn't survive while in V a node n survives. Therefore $p_{occur}(V \setminus \{n\}) = \frac{p_{occur}(V)}{1-dp(n)} \times dp(n)$.

$$X_{s11} = \{n_2, n_3\}, \quad \text{and} \quad p_{occur}(X_{s11}) = (1 - 0.6)(1 - 0.4)(0.7) = 0.168;$$

$$X_{s12} = \{n_1, n_3\}, \quad \text{and} \quad p_{occur}(X_{s12}) = (1 - 0.7)(1 - 0.4)(0.6) = 0.108;$$

$$X_{s13} = \{n_1, n_2\}, \quad \text{and} \quad p_{occur}(X_{s13}) = (1 - 0.7)(1 - 0.6)(0.4) = 0.048.$$

As $\alpha = 1$, we use the set X_{s11} to generate subsets in the next level. $X_{s21} = \{n_2\}$ and $X_{s22} = \{n_3\}$ are both removed because they are invalid. The search terminates because no more valid subsets can be created. The survivability estimate computed by Algorithm *SF*3 for the deployment is $SF3(\mathcal{M}, V, dp, \mu, \alpha, K_b, K_r) = 0.396$. Hence Algorithm *SF*3 lower bounds $SF(\mu)$, which is 0.468, as shown in Example 4.

4.3. A disjoint based Algorithm SF4

Algorithm *SF*4 calculates survival probabilities while relying on the requirement that every agent must survive somewhere in the network, and using information on the locations of each agent. For each agent $a_i \in \mathcal{M}$ in the multi-agent application, let $Loc(a_i) = \{n_1^i, \ldots, n_k^i\}$ be the set of nodes where a_i is located. Let E_j^i be the event that the node n_j^i will survive. Then the event that at least one copy of a_i will keep functioning is denoted by $E^i = E_1^i \vee \cdots \vee E_k^i$. The probability of the event E^i can be computed by

$$P(E^i) = 1 - dp(n_1^i)dp(n_2^i)\ldots dp(n_k^i).$$

We can now define the event that a MAS deployment μ will survive:

$$E(\mathcal{M}, V, dp, \mu) = (E_1^1 \vee \cdots \vee E_{k_1}^1) \wedge \cdots \wedge (E_1^{|\mathcal{M}|} \vee \cdots \vee E_{k_{|\mathcal{M}|}}^{|\mathcal{M}|}).$$

The probability of the event $E(\mathcal{M}, V, dp, \mu)$ represents the survivability of the deployment μ. Unfortunately, the E^is are not mutually exclusive. However, Algorithm *SF*4 computes a lower bound of $E(\mu)$ by assuming that the events $E^1, E^2, \ldots, E^{|\mathcal{M}|}$ are pairwise disjoint. The *SF*4 algorithm is defined by the following formula:

$$SF4(\mathcal{M}, V, dp, \mu) = P(E^1)P(E^2)\cdots P(E^{|\mathcal{M}|}) = (1 - dp(n_1^1)\cdots dp(n_k^1)) \times \cdots \times (1 - dp(n_1^{|\mathcal{M}|})\cdots dp(n_r^{|\mathcal{M}|})).$$

Proposition 11. *Algorithm SF4 provides a lower bound on SF(μ).*

The following example illustrates the operation of Algorithm *SF*4.

Example 6. Consider the updated deployment μ' of Example 4. Agents a_1 and a_2 are located at nodes $V = \{n_1, n_2\}$ and $V' = \{n_1, n_3\}$, respectively. Thus

$$P(E^1) = 1 - dp(n_1)dp(n_2) = 1 - 0.42 = 0.58;$$

$$P(E^2) = 1 - dp(n_1)dp(n_3) = 1 - 0.28 = 0.72.$$

So the survivability of μ is computed by $SF4(\mathcal{M}, V, dp, \mu) = P(E^1)P(E^2) = 0.4176$.

Algorithm *SF*4 returns a lower bound on $SF(\mu)$ (i.e., 0.468). In this example, compared with the value returned by Algorithm *SF*3 (0.396, see Example 5), Algorithm *SF*4 provides the better solution. However, as shown later in the section on experiments, there are some other cases where Algorithm *SF*4 returns lower survivabilities than Algorithm *SF*3.

4.4. A group based Algorithm SF4$_g$

Algorithm *SF*4 computes each agent's survival probability and then returns the product of these survival probabilities. *If no node contains more than one agent, then Algorithm SF*4 *returns the exact answer.* However, in general, when the number of agents is large and there is a large number of nodes in which many agents are located, Algorithm *SF*4 may return a very low approximation ratio. To improve this, if there are agents in a deployment that coexist in various nodes, we would consider these agents as a group and compute the group's survivability. We divide all agents into several such groups, and then take the product of the survival probabilities of all groups as the survivability of the deployment. An intuitive way to group agents is to consider an agent a that has the lowest survivability. We group a with other agents that have the most common nodes with it. When we compute the survivability of each agent group, we use Algorithm $SF1_a$. As Algorithm $SF1_a$ takes exponential time in the number of agents, we limit the size of each group.

Algorithm 4 *(Algorithm $SF4_g(\mathcal{M}, V, dp, \mu, s)$).*

```
(∗ Input: the number of agents in one group, $s$ ∗)
(∗ Output: a lower bound on $SF(\mu)$ ∗)
```

```
(1) $agents = \mathcal{M}$, $surv = 1$;
(2) for each agent $a_i \in agents$, do
        $P(E^i) = 1 - \prod_{n \in Loc(a_i)} dp(n)$;
(3) while ($agents \neq NULL$), do
    • $a_i = \arg\min_{a_i \in \mathcal{M}} P(E^i)$; (* choose the agent with the lowest survivability *)
    • $A' = group(a_i, s, \mu, V, agents)$;
      (* group at most $s-1$ agents who have the most common locations with $a_i$ into one group $A'$ *)
    • $value = SF1_a(A', V, dp, \mu)$; (* use Algorithm $SF1_a$ to compute the survivability of $A'$ *)
    • $surv = surv \times value$;
    • $agents = agents \setminus A'$.
(4) return $surv$.
```

In this algorithm, the function $group(a, s, \mu, V, agents)$ takes as input, an agent a, the predefined number of members in one group s, the deployment μ, a set of agents denoted *agents* and a set of nodes V. For each agent, it computes the number of common locations with agent a, selects the first $s-1$ agents who have the most common nodes (if two agents have the same number of common nodes it chooses one of them arbitrarily), groups these $s-1$ agents together with agent a into one group A', and then returns A'.

The following example demonstrates the grouping idea applied in Algorithm $SF4_g$.

Example 7. Consider a network with six nodes $V = \{n_1, n_2, n_3, n_4, n_5, n_6\}$ and a multi-agent application $\mathcal{M} = \{a_1, a_2, a_3, a_4\}$. Suppose the disconnect probabilities are:

$$dp(n_1) = 0.7, \quad dp(n_2) = 0.6, \quad dp(n_3) = 0.4, \quad dp(n_4) = 0.3, \quad dp(n_5) = 0.2, \quad dp(n_6) = 0.1.$$

Consider the current deployment μ:

$$\mu(n_1) = \{a_1, a_2\}, \quad \mu(n_2) = \{a_1\}, \quad \mu(n_3) = \{a_2\}, \quad \mu(n_4) = \{a_3\}, \quad \mu(n_5) = \{a_3, a_4\}, \quad \mu(n_6) = \{a_4\}.$$

We set the number of agents per group to 2. Algorithm $SF4_g$ first computes the survivability of each agent based on the nodes in which the agent is located:

$$P(E^1) = 1 - dp(n_1)dp(n_2) = 0.58, \quad P(E^2) = 1 - dp(n_1)dp(n_3) = 0.72,$$
$$P(E^3) = 1 - dp(n_4)dp(n_5) = 0.94, \quad P(E^4) = 1 - dp(n_5)dp(n_6) = 0.98.$$

Since agent a_1 is most likely to fail, we select a_1 to form the first group g_1. We then group agent a_2 and a_1 together since a_2 has the most number of common nodes with agent a_1. Thus we have $g_1 = \{a_1, a_2\}$. Algorithm $SF1_a$ is applied to group g_1 to compute its survivability:

$$\begin{aligned} SF(g_1, V, dp, \mu) &= SF1_a(g_1, V, dp, \mu) = 1 - \big(Pr(A_{a_1}) + Pr(A_{a_2})\big) + Pr(A_{a_1} \wedge A_{a_2}) \\ &= 1 - \big(dp(n_1)dp(n_2) + dp(n_1)dp(n_3)\big) + dp(n_1)dp(n_2)dp(n_3) = 0.468. \end{aligned}$$

Similarly, the remaining agents, a_3 and a_4, form the second group $g_2 = \{a_3, a_4\}$.

$$\begin{aligned} SF(g_2, V, dp, \mu) &= SF1_a(g_2, V, dp, \mu) = 1 - \big(Pr(A_{a_3}) + Pr(A_{a_4})\big) + Pr(A_{a_3} \wedge A_{a_4}) \\ &= 1 - \big(dp(n_4)dp(n_5) + dp(n_5)dp(n_6)\big) + dp(n_4)dp(n_5)dp(n_6) = 0.914. \end{aligned}$$

Thus, the estimated survivability given by the algorithm is: $SF4_g(\mu, V, dp, \mu, 2) = SF(g_1, V, dp, \mu)SF(g_2, V, dp, \mu) = 0.42775$.

Using Algorithm $SF1_a$ on $\mathcal{M}$, we know the survivability of μ is $SF(\mu) = 0.42775$. Clearly, in this case, Algorithm $SF4_g$ returns the actual survivability since there is no overlap whatsoever between the locations of two groups ($g_1 = \{a_1, a_2\}$ and $g_2 = \{a_3, a_4\}$). However, this may not necessarily happen with other deployments.

4.5. A split Algorithm SF5

Given a specific node, $n \in V$, we can consider two possible disjoint events. The first, E_1, is the event that the network survives given that node n remains connected. Alternatively, E_2 is the event that the network survives given that node n becomes disconnected. If n remains connected, all the agents that are deployed on it survive. Thus the survivability of the network, in this case, depends on the survivability of the rest of the agents, which are not located on n. If n is disconnected, then the survivability of the network depends on the rest of the nodes, i.e., $V \setminus \{n\}$. The survivability of the original network is thus $(1 - dp(n))Pr(E_1) + dp(n)Pr(E_2)$. In both cases, the problem of computing the survival probability is smaller than the original problem. This leads to a recursive approach for solving the problem. The subproblems usually become even smaller when we remove the irrelevant agents according to the idea presented in [27]. There are several stopping rules that are specified in the first three lines of the pseudocode shown below. The first two rules refer to situations in which it is possible to compute the exact survival probability of the future network. The third stopping rule has to do with future networks that have a very small survival probability (computing through recursion using p; $p = 1$ the first time Algorithm *SF*5 is called). For these very low probability future networks, Algorithm *SF*4 is applied to obtain a lower bound of the survivability.

Algorithm 5 *(Algorithm* $SF5(\mathcal{M}, V, dp, \mu, p, \epsilon)$*).*

```
(∗ Input: (1) survivability of the known nodes during split $p$; initially $p = 1$ ∗)
(∗        (2) a predefined threshold $\epsilon$ ∗)
(∗ Output:    a lower bound on $SF(\mu)$ ∗)

(1) if $\mathcal{M} = \emptyset$, return 1;
    else, if the agents of $\mathcal{M}$ are located on disjoint sets of nodes,
        then return $\prod_{a\in\mathcal{M}}(1 - \prod_{a\in\mu(n)} dp(n))$;
    else, if $p < \epsilon$,
        then return $SF4(\mathcal{M}, V, dp, \mu)$.
    else, choose a node $n \in V$ with the largest set of agents,
    (a) $V'' = V' = V \setminus \{n\}$;
    (b) $\mu' = \mu$; $\mathcal{M}' = \mathcal{M} \setminus \{a \mid a \in \mu(n)\}$;
    (c) get rid of irrelevant agents in $\mathcal{M}$ and $\mathcal{M}'$ according to the idea in [27];
    (d) adjust $\mu$ and $V'$ w.r.t. $\mathcal{M}$ and $\mu'$ and $V''$ w.r.t $\mathcal{M}'$;
    (e) return $dp(n) \times SF5(\mathcal{M}, V', dp, \mu, dp(n)p, \epsilon) + (1 - dp(n)) \times SF5(\mathcal{M}', V'', dp, \mu', (1 - dp(n))p, \epsilon)$.
```

Proposition 12. *Algorithm SF5 yields a lower bound on SF*(μ)*.*

Proof. Algorithm *SF*5 is recursive, with three termination conditions. If the termination conditions are not met, the algorithm generates a (smaller) subproblem; if the algorithm terminates, there are three possible cases as follows.

If the algorithm terminates by the *first* termination condition, i.e., $\mathcal{M} = \emptyset$ (line 1 of the pseudocode in Algorithm 5), the subnetwork of the subproblem does not contain any agents. Thus we cannot simplify the problem any further. The result returned by the algorithm is the exact solution for the original problem.

If the *second* termination condition applies, i.e., the agents of $\mathcal{M}$ are located on disjoint sets of nodes, return $\prod_{a\in\mathcal{M}}(1 - \prod_{a\in\mu(n)} dp(n))$ (lines 2, 3 of the pseudocode in Algorithm 5), the subproblem is easily computed by multiplying each agent's survivability. Since the agents in this condition are located in disjointed nodes, the solution is the exact same solution as for the original problem.

Finally, if Algorithm *SF*5 terminates by the *third* condition, i.e., if $p < \epsilon$, then return $SF4(\mathcal{M}, V, dp, \mu)$ (lines 4, 5 of the pseudocode in Algorithm 5). Since Algorithm *SF*4 provides a lower bound on $SF(\mu)$, the result returned by Algorithm *SF*4 is a lower bound for the subproblem of Algorithm *SF*5.

In conclusion, Algorithm *SF*5 also provides a lower bound on $SF(\mu)$. □

The following example illustrates how Algorithm *SF*5 works.

Example 8. Consider the network and the updated deployment of Example 4, where $\mu(n_1) = \{a_1, a_2\}$, $\mu(n_2) = \{a_1\}$, and $\mu(n_3) = \{a_2\}$, and $\epsilon = 0.001$. The heuristics we use to choose the node to split is the number of agents deployed on that node. Since node n_1 has the largest number of agents of all the nodes, n_1 is selected. There are two cases which refer to the splitting operation w.r.t. n_1: The first event E_1 is where the network will survive given that node n_1 will remain connected. In this case we have:

$$V_1 = \{n_2, n_3\}, \quad \mathcal{M}_1 = \emptyset, \quad \forall n \in V_1, \quad \mu_1(n) = \emptyset, \quad p_1 = 1 - dp(n_1) = 0.3;$$

The second event E_2 is where the network will survive given that node n_1 will be disconnected. In this case we have:

$$V_2 = \{n_2, n_3\}, \quad \mathcal{M}_2 = \{a_1, a_2\}, \quad \mu_2(n_2) = \{a_1\}, \quad \mu_2(n_3) = \{a_2\}, \quad p_2 = dp(n_1) = 0.7;$$

We now call Algorithm *SF*5 with the updated parameters.

As $\mathcal{M}_1 = \emptyset$, we have $Pr(E_1) = 1$. In $\mathcal{M}_2$, the agents a_1 and a_2 are located on disjoint nodes, thus, we return

$$Pr(E_2) = \big(1 - dp(n_1)\big)\big(1 - dp(n_2)\big) = (1 - 0.4)(1 - 0.6) = 0.24;$$

Therefore, we stop the split operation and return the survivability of the original deployment by:

$$SF5(\mathcal{M}, V, dp, \mu, 1, 0.001) = p_1 \times Pr(E_1) + p_2 \times Pr(E_2) = 0.468.$$

Note that in this case, Algorithm *SF*5 outputs the exact survivability (0.468, as shown in Example 4). However, this is not necessarily true for other cases.

5. Experiments

The survivability algorithms we shall compare in this section are:

- *anytime algorithm SF*2: where the threshold of the approximation ratio is set to 0.9 and the time limit on the main part of the algorithm is set to 5 seconds;
- *tree-based algorithm SF*3, where the constant α is set to the number of nodes in the network, and the constants K_b and K_r are set to 20 and 6, respectively;
- *disjoint based algorithm SF*4;
- *group algorithm* $SF4_g$, where the number of agents in each group is set to 4;
- *split algorithm SF*5, where the threshold ϵ in the stopping rules is predefined as 0.005.

The performance measures we used are: (1) computation time, and (2) solution quality. We evaluate the solution quality as follows.

- Exact optimal solutions: For small cases where the number of nodes or the number of agents are small (less than 16), we obtain the exact survivability S_E by using the naive algorithm $SF1_n$ when there are more agents than nodes, or by using $SF1_a$ when there are more nodes than agents. The solutions provided by the heuristic algorithms S_H are then compared. The *solution quality* (or *approximation ratio*), is computed as: $\frac{S_H}{S_E}$.
- Upper bounds on optimal solutions: For large cases where computing optimal solutions is not feasible, we compute the upper bounds of the exact survivabilities *UB* using the upper bound algorithm and compare them with the values S_H returned by the heuristics. Thus, the *solution quality* (or *approximation ratio*) is evaluated by: $\frac{S_H}{UB}$.

We considered various experimental settings. In this paper, we consider instances taken from a (fictitious) company that uses local servers, personal computers, and some web servers to locate and run multi-agent applications. As we know, web servers and personal computers have high probabilities of going down, while local servers usually have lower disconnect probabilities. In the next section, we describe the variations of the settings we used in our experiments. We use the term *number ratio* to refer to the ratio of the number of agents to the number of nodes. *Space ratio* describes the ratio of the total amount of resources available on the nodes to the total resource requirements of the agents. In addition, the *problem size* is the sum of the number of agents and nodes in the settings.

5.1. Environmental settings

We used various environmental settings in the experiments. Suppose a multi-agent application $\mathcal{M}$ includes many agents but only a relatively small number of servers (or nodes) is available. We set the number ratio of agents and nodes to 5/3. We then considered the following two environments:

s1: A network consisting of a small number of web servers N_w which constitute 30% of the involved servers, and many local servers N_l, which constitute 70% of the involved servers. The disconnect probabilities of the web servers are very high—above 0.9 (i.e., $dp(n) \geqslant 0.9$, $\forall n \in N_w$), while the disconnect probabilities of the local servers are very low—below 0.1 (i.e., $dp(n) \leqslant 0.1$, $\forall n \in N_l$). The space ratio of nodes and agents is between 2 and 3.

s3: A network consisting of local servers only. Suppose some of these servers are new, while the others are old. The disconnect probabilities of these servers are between 0 and 0.4, where higher disconnect probabilities have a higher probability to appear (there are more older computers than new ones). The space ratio of nodes and agents is 4.

Consider another multi-agent application $\mathcal{M}'$ which consists of a small number of agents. The company intends to deploy $\mathcal{M}'$ on many personal computers and local servers since the available resources on each server or PC are limited. We set the number ratio of agents to nodes at 3/5. The following environments are specified:

s2: Personal computers (30%) are employed, with disconnect probabilities over 0.9; they also use local servers (70%) which have low disconnect probabilities (less than 0.1). The space ratio of nodes to agents is around 2–3.

s4: Only local servers of different agents are used to host $\mathcal{M}'$. The disconnect probability of the servers is distributed as in setting s3. The space ratio of nodes to agents is around 4.

We apply an existing MAS application from the IMPACT system [40] with 31 agents to determine the resource distribution of agents (in the range of 0 to 250 KB) in our experiments.[10] We use the environments s1–s4 described above to test the survivability algorithms.

[10] We do not distinguish the criticality between agents in a multi-agent system when computing its survivability. Therefore, other factor such as roles (or workloads) of different agents does not play a role in the experiments, and thus not reported here.

Table 1
Experiment 1: Upper bounds and approximation ratios of the different algorithms with setting $s1$.

Problem size	Deploy method	Upper bound	Anytime ($SF2$)	Tree-based ($SF3$)	Disjoint ($SF4$)	Split ($SF5$)	Group-based ($SF4_g$)
$n18, a30$	node-based	0.435412	0.951475	0.998919	0.879377	0.999515	0.948644
	agent-based	0.349391	0.930565	0.972725	0.879446	0.973295	0.947005
	random-based	0.002281	–	–	–	–	–
$n24, a40$	node-based	0.355659	0.85197	0.964037	0.836856	0.965813	0.923682
	agent-based	0.306698	0.861461	0.881713	0.796948	0.889531	0.859259
	random-based	0.000166	–	–	–	–	–
$n30, a50$	node-based	0.429062	0.899324	0.937145	0.775934	0.939865	0.893412
	agent-based	0.331821	0.811792	0.851059	0.731787	0.852492	0.827303
	random-based	0.000006	–	–	–	–	–

5.2. Agent deployment methods

The method used to generate deployment is important since different survivability algorithms may work well with different types of deployments. In this paper, we generate deployments by the heuristics proposed in [27], namely node-based and agent-based heuristics. In addition, we use a random-based method to represent other possible deployments. The deployment methods work as follows.

Node-based: This heuristic is based on the knapsack problem. We first sort nodes in ascending order according to their disconnect probabilities. We then place agents, starting from the one with the smallest ID in our implementation, on the sorted nodes starting from the node with the lowest *dp*. We put as many agents as possible on this node, then go to nodes with the second lowest *dp* and so on.

Agent-based: This is based on the idea that we should first deal with agents with high resource requirements. We sort agents in ascending order according to resource requirements, deploy the first agent, then choose the agent with the second highest resource requirement, and so on until no more space is left for placing agents. When we deploy an agent, we always choose the node with the lowest dp of the nodes capable of storing the said agent.

Random-based: First we randomly choose a node, and then randomly select and place agents on it, subject to space constraints. We make sure the deployment uses all the available resources on the nodes.

In the experiments, we also wanted to investigate the performance of the survivability algorithms on different deployments returned by various deployment methods, and to check whether the algorithms have preferences for particular deployments.

5.3. Experimental results

We are now ready to present the experimental results. All the algorithms in this paper were implemented on a Linux PC running on a 1000 MHz CPU machine with 512 MB RAM. *Running times of all algorithms are reported in microseconds*. Every recorded observation was averaged over 50 runs. The algorithms were compared on various deployments (node-based, agent-based, and random-based) with different environment settings s1–s4. In all the experiments, we varied the problem size, i.e., the total number of agents and nodes. We present the results of the following experiments: *Experiment 1* was carried out in setting $s1$; *Experiment 2* in setting $s2$; *Experiment 3* in setting $s3$; *Experiment 4* in setting $s4$; *Experiment 5* compared different algorithms in setting $s1$ but with a larger space ratio.

In order to ensure that the survivability computed by the upper bound algorithm is close enough to the actual value, for each deployment, we estimated its survivability by simulating the node failures on the network a thousand times. The results show that the upper bounds are pretty close to the simulation survivabilities. The average relative error, e.g., $\frac{\textit{upper bound-simulation result}}{\textit{simulation result}}$, is within 0.5%.

The *approximation ratio of the algorithm* reported in the experimental result is the solution quality described on page 450, which is computed by either $\frac{S_H}{S_E}$ or $\frac{S_H}{UB}$, depending on the problem size.

Experiment 1

In Experiment 1, we ran and compared five algorithms in setting $s1$ where the space ratio of nodes to agents was between 2 and 3 and the disconnect probabilities were distributed either in 0–0.1 or in 0.9–1. The problem size varied from 48, 64 to 80. Table 1 illustrates the results of upper bounds on the survivability and approximation ratios by the different algorithms, where $n18, a30$ refers to a MAS of 30 agents deployed over 18 nodes.

As can be seen from the upper bounds in Table 1, the deployments achieved by the node-based deployment method have a higher survivability than the agent-based deployments. Undoubtedly, when the disconnect probabilities of the nodes vary dramatically, the MAS is better off if the highly surviving nodes are considered first when deploying agents. Therefore, it is not surprising that the node-based deployments result in the best survivabilities of all the methods, and the random based deployment method returns very poor survivabilities (all below 0.005 according to the upper bounds in the table).

Table 2
Experiment 1: Computation time (in microseconds) using the different algorithms with setting $s1$.

Problem size	Deploy method	Anytime ($SF2$)	Tree-based ($SF3$)	Disjoint-based ($SF4$)	Split ($SF5$)	Group-based ($SF4_g$)
$n18, a30$	node-based	65179	5999	7	26094	927
	agent-based	75635	3874	5	22075	710
$n24, a40$	node-based	1237829	2877	9	55172	1527
	agent-based	1237632	975	7	51954	1066
$n30, a50$	node-based	5105916	10156	11	73187	2231
	agent-based	4302199	5039	9	71481	1707

Table 3
ANOVA on the results of five algorithms in Experiment 1 with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	0.16483	4	0.04121	10.79146	2.9689E–07	2.46749
Residual	0.36275	95	0.00382			
Total	0.52758	99				

Table 4
ANOVA on the results of $SF3$ and $SF5$ in Experiment 1 with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	1.0266E–06	1	1.0266E–06	2.9647	0.09323	4.09817
Residual	1.3158E–05	38	3.4627E–07			
Total	1.4185E–05	39				

In Table 1, we exclude the random based deployments (due to their very low survivabilities) and discuss the approximation ratios returned by the various algorithms only on agent-based and node-based deployments. Of all the heuristics, the tree based algorithm, $SF3$, and the split algorithm, $SF5$, return the best solutions. $SF5$ attains the best result, although the difference between its results and those of $SF3$ is very small. $SF3$ always achieves higher accuracy than the disjoint based algorithm $SF4$, the anytime algorithm $SF2$ and the group algorithm $SF4_g$. The reason for this is twofold.

(1) First, when searching for the valid future networks, in each level of the tree, $SF3$ always keeps the networks which have higher survivabilities, and removes those more likely to fail. Thus, in the $s1$ setting where the disconnect probabilities are either very high or very low, $SF3$ will select the valid future networks where most nodes have low disconnect probabilities, i.e., $dp < 0.1$. Since the valid future networks with low survival nodes ($dp > 0.9$) do not greatly contribute to the survivability of the deployment, $SF3$ can make good approximations.
(2) Second, the space ratio of nodes to agents is small (i.e., 2–3) in the $s1$ setting, which implies that since the nodes cannot accommodate many agents due to resource constraints, the number of valid future networks is limited to a relatively small number. As $SF3$ keeps the fixed α best networks in each level of the tree search, it is very likely that $SF3$ will keep most of the "important" valid future networks which have a relatively high survivability. Thus, $SF3$ works very well with setting $s1$.

Statistical significance. In order to discover whether there are significant differences in the performances of the algorithms, we performed a *One-way Analysis of Variance*, or one-way ANOVA[11] [12], on the solution of each round returned by the different algorithms. Table 3 shows the results of the ANOVA applied to the data achieved in setting $s1$ on the agent-based deployments when the problem size is 64. The level of significance required in ANOVA is set to 0.05. In the table, the calculated F value (F = 10.79146) exceeds the critical value of F (2.46749) and the probability (p-value) that the calculated F value would be obtained by change is nearly zero. Therefore, the difference in performance of the five algorithms is significant. We then performed the ANOVA only on the tree based algorithm $SF3$ and the split algorithm $SF5$. Since the computed F value (F = 2.9647) is smaller than the critical value of F (4.09817) as shown in Table 4, we can conclude that there is no significant difference between the performance of $SF3$ and $SF5$.

[11] In one-way ANOVA, the number of degrees of freedom (df) associated with "between algorithms" is one less than the number of algorithms; the df for "residual" is the total number of samples of the algorithms minus the total number of algorithms. The F value is simply the ratio of the variance estimates of "between algorithms" and "residual". The critical values (F crit) are presented in an F table. Using the F, we can compute the p-value, which is the probability of the obtained result occurring due to chance. For a more detailed discussion on one-way ANOVA, see [12].

Table 5
Experiment 2: Upper bounds and approximation ratios of the different algorithms with setting *s*2.

Problem size	Deploy method	Upper bound	Anytime (*SF*2)	Tree-based (*SF*3)	Disjoint (*SF*4)	Split (*SF*5)	Group-based ($SF4_g$)
*n*30, *a*18	node-based	0.520557	0.928353	0.990334	0.974706	0.99961	0.989189
	agent-based	0.410592	0.939275	0.996562	0.973966	0.999569	0.998401
	random-based	0.001406	–	–	–	–	–
*n*40, *a*24	node-based	0.436522	0.880772	0.980832	0.952544	0.98856	0.979202
	agent-based	0.342072	0.907723	0.968986	0.941974	0.991491	0.988068
	random-based	0.001808	–	–	–	–	–
*n*50, *a*30	node-based	0.398411	0.923707	0.981228	0.958711	0.973521	0.96628
	agent-based	0.317906	0.910653	0.972855	0.900914	0.976648	0.967943
	random-based	0.001918	–	–	–	–	–

Table 6
Experiment 2: Computation time (in microseconds) using the different algorithms with setting *s*2.

Problem size	Deploy method	Anytime (*SF*2)	Tree-based (*SF*3)	Disjoint-based (*SF*4)	Split (*SF*5)	Group-based ($SF4_g$)
*n*30, *a*18	node-based	5904	26546	8	19842	1361
	agent-based	29781	1325	7	14043	586
*n*40, *a*24	node-based	55806	176935	11	39496	2111
	agent-based	225411	1183	8	32944	998
*n*50, *a*30	node-based	364816	1109754	15	86811	3096
	agent-based	1822433	11842	11	74323	1676

Table 7
ANOVA on the results of *SF*3 and *SF*5 in Experiment 2 with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	0.00048	1	0.00048	11.54003	0.0012	3.99092
Residual	0.00265	64	4.1477E–05			
Total	0.00313	65				

Computation time. Table 2 shows the computation time taken by the different algorithms. From the results, we can see that *SF*3 needs much less computation time than the split algorithm and the anytime algorithm on both node and agent-based deployments. The disjoint based algorithm is the fastest of all, taking only several microseconds to compute the survivability of a given deployment. The time needed by the group algorithm $SF4_g$ is close to that of the tree based algorithm *SF*3.

Conclusion. Overall, in experimental setting *s*1, when taking both the approximation ratio and the computation time into account, the tree based algorithm *SF*3 outperforms the other algorithms.

Experiment 2

Tables 5 and 6 present the results of the performances of the different algorithms in experimental setting *s*2 in which there are more nodes than agents (with a ratio of 5/3), the space ratio of nodes to agents is 2–3, and the *dp*'s are distributed dramatically. Concerning the upper bounds of the deployments returned by the different types of methods, the node-based method results in the highest survivability. Again, the random based method does not work well in this setting with dramatically varying dp's over the network.

In terms of solution quality, the split algorithm *SF*5 is the best, followed by the tree based algorithm *SF*3, which outperforms the group algorithm $SF4_g$ both on the node-based deployments with problem sizes of 48, 64, 80, and on the agent-based deployments with a size of 80. The disjoint algorithm *SF*4 gives a pretty good approximation (with a ratio of over 0.9), regardless of the fact that it returns the poorest measurements of all algorithms. The performance of both *SF*4 and $SF4_g$ is significantly better than in Experiment 1. In the deployments achieved with setting *s*2, the average number of agents located on each node is smaller compared to setting *s*1 since there are more nodes than agents in *s*2, but the space ratio is the same as in *s*1. As there is less overlap in the locations of agents, *SF*4 and $SF4_g$ work better in this experiment than they did in Experiment 1.

Statistical significance. In addition, we performed a statistical significance test for the algorithms *SF*3 and *SF*5. Table 7 shows that the calculated F value ($F = 11.54003$) is greater than the critical value of F (3.99092) with a very small p value ($p = 0.0012$). Hence, we conclude that in this setting, the performance of algorithms *SF*3 and *SF*5 in terms of solution quality are significantly different.

Table 8
Experiment 3: Upper bounds on the actual survivability of the different deployments with setting s3.

	$n12$, $a20$	$n18$, $a30$	$n24$, $a40$	$n30$, $a50$	$n36$, $a60$
node-based	0.928231	0.889023	0.876090	0.858844	0.813199
agent-based	0.935944	0.867488	0.836907	0.831710	0.803683
random-based	0.657132	0.474367	0.349565	0.302506	0.186378

Computation time. In terms of computation time taken by different heuristics on the agent-based deployments, the results are similar to those in Experiment 1, i.e., the split algorithm *SF*5 and the anytime algorithm *SF*2 take much more time than the other three heuristics. If we look at the time taken for computing the node-based deployments, it is clear that the tree based heuristic *SF*3 needs the longest time. As we have shown in Section 4.2, the complexity to compute *SF*3 is dependent on $\alpha|V|^2\log(\alpha|V|)$. Therefore, although the problem size remains the same in setting $s2$ as in $s1$, *SF*3 needs longer time to terminate compared to $s1$ since the ratio of nodes to agents is higher. However, note that this is not the case for *SF*3 on the agent-based deployments, where *SF*3 completes the search fast, but the survivabilities it returns are comparable with those on the node-based deployments, as depicted in Tables 5 and 6. Apparently with setting $s2$, when applying *SF*3 on the agent-based deployments, the number of valid future networks in each level converges fast with the tree search. Consequently, *SF*3 may terminate early. Thus, the results suggest that in settings like $s2$, the tree based algorithm *SF*3 leads to better performance in terms of computation time for the agent-based deployments compared to the node-based ones.

Conclusion. Overall, in experimental setting $s2$, the split algorithm *SF*5 is the preferred algorithm if both the solution quality and the running time are taken into account.

Experiment 3

Experiment 3 was performed in setting $s3$, where the disconnect probability of nodes was distributed between 0–0.4 and higher disconnect probabilities had a higher probability to appear. From the results in Table 8, we notice that this type of disconnect probability distribution immediately increases the survivability of the deployments, compared with the upper bounds shown in Experiments 1 and 2. Even randomly deploying agents could result in better deployments with higher survivability than those in Experiments 1 and 2. The node-based method still seems to find better surviving deployments than the other two methods.

Figs. 1 and 2 present the approximation ratios and the computation time of the various algorithms, respectively. In both figures, the x-axis represents the problem size, varying from 32 to 98 in steps of 16. Note that we did not include the results of the tree based algorithm *SF*3 since its approximation ratio was much lower (below 0.8) than the others in setting $s3$. Since in $s3$ the disconnect probabilities of the nodes do not vary dramatically, the values of all the valid future networks do not greatly vary. Therefore, the bad approximation of *SF*3 is due to the fact that it excludes the survivabilities of many networks which also significantly contribute to the overall survivability.

Fig. 1 demonstrates the advantage of the split heuristic *SF*5, which gives the best approximation ratio no matter what kind of deployments it employs. As described in Section 4.5, *SF*5 is a recursive algorithm and after each recursion the subproblem becomes smaller. Moreover, when a node is chosen during computation, the whole network could be updated by removing the irrelevant agents, which further reduces the size of the subproblems. The algorithm terminates only either when the future subnetwork (or subproblem) can be accurately computed, or when the survivability of the future subnetwork becomes small enough to be roughly estimated—the threshold is set to 0.005 in the experiments. Thus it is not surprising that *SF*5 yields a very good approximation. According to the experiments in this paper the split algorithm *SF*5 always provides a good approximation ratio regardless of the environment settings.

Furthermore, all the algorithms achieve high approximation ratios (over 0.96) on the node-based deployments. As for the agent-based deployments, all the algorithms return approximation ratios of over 0.90 with a problem size no larger than 80. Only the ratio of *SF*4 drops to 0.86 when the problem size rises to 96. All the algorithms excluding the anytime algorithm reach above a 95% accuracy on the random-based deployments. Seemingly the *SF*4 and the $SF4_g$ do significantly better on node-based deployments than on agent-based ones.

Statistical significance. Table 9 shows the ANOVA test result on algorithms *SF*2, $SF4_g$ and *SF*5. The calculated F value (20.17751) is greater than the critical value of F (3.96347). Thus the performances of these three algorithms are significantly different. However, Table 10 implies that there may not be a great difference between *SF*2 and *SF*5, since the calculated F value (2.45835) is smaller than the critical value (3.96347).

Computation time. As far as the computation time is concerned (shown in Fig. 2 in a logarithmic scale), the anytime algorithm is the time consuming one. The disjoint-based *SF*4 is the fastest algorithm of all the algorithms. The three graphs show that the computation times of the disjoint based, the group based, and the split algorithms are barely affected by the deployment methods they employ. However, the anytime algorithm converges much faster on the node-based deployments than the agent-based deployments—the computation time taken on the latter is approximately 100 times more when the problem size is over 48.

Conclusion. In setting $s3$, the split algorithm *SF*5 and the anytime algorithm *SF*2 outperform other algorithms in terms of solution quality. However, *SF*5 is preferable to *SF*2 since *SF*5 converges to a solution much faster than *SF*2.

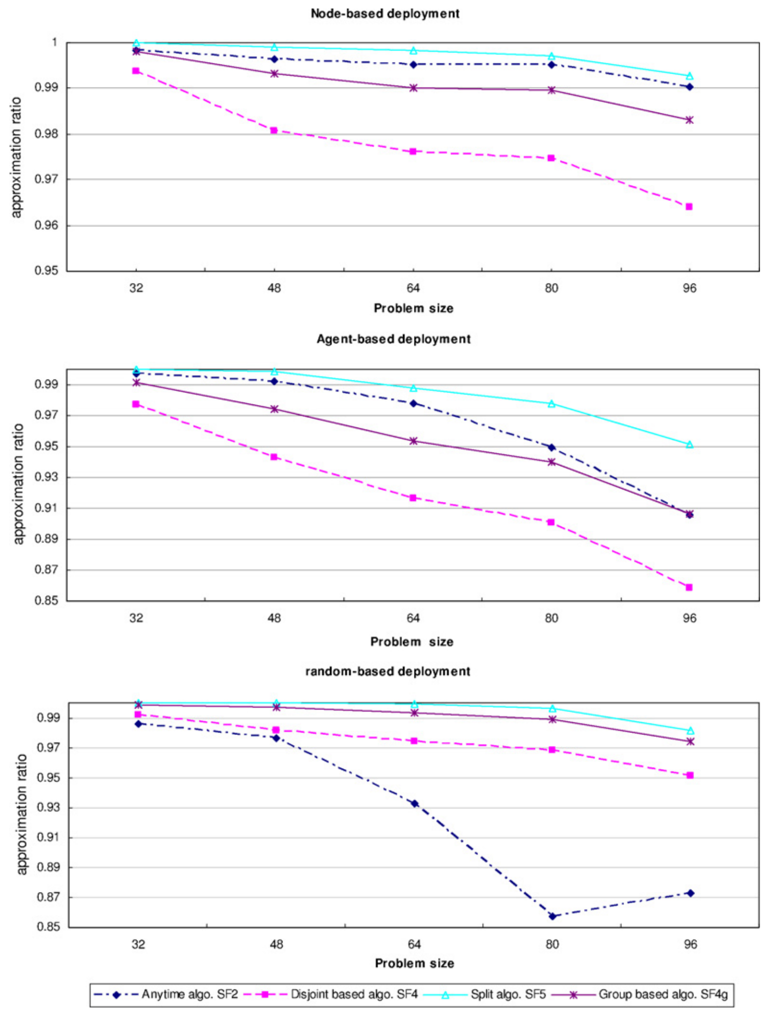


Fig. 1. Experiment 3: Approximation ratios of the different algorithms with setting *s*3. The *x*-axis represents the problem size and the y-axis represents the approximation ratio.

Experiment 4

Experiment 4 was carried out with environment setting *s*4, where there are more nodes than agents (with a ratio of 5/3), and dp's are distributed as in *s*3. Table 11 contains the results of the upper bounds and approximation ratios by the various algorithms. With this setting, even the random based deployments achieve survivabilities over 0.5 for problem sizes below 80. Using the agent and node-based methods to deploy agents both result in deployments with upper survivability bounds exceeding 0.97. Moreover, for the first time, the agent-based method seems to result in better deployments than the node-based method, although the differences are quite small (within 0.01). Since the upper bounds are high, we can assume that the real survivability of the deployments are high—which suggests that in the deployments, every agent is deployed on most of nodes in the network; and/or at least one of the valid future networks has a very high survivability.

As far as the accuracy of the various algorithms is concerned, all the algorithms exhibited excellent accuracies which always exceed 0.97. In particular, the disjoint-based algorithm *SF*4, the split and the group algorithms (*SF*5 and $SF4_g$) were able to always achieve approximation ratios over 0.998. This is due to the fact that when there are more nodes than agents, the resources available in each node decreases compared with that in Experiment 3. Thus most of the agents are disjoint from the others w.r.t. their locations, which enables the algorithms, especially the disjoint and the group algorithms, to perform very well.

Statistical significance. Since all four algorithms displayed a very good solution quality, we performed the ANOVA test in order to assess statistical significance. The test revealed a significant difference as depicted in the results reported in Table 12, where the calculated F value (10.9514) is greater than the critical value of F (2.6625). We did the test again on the same data but only for the anytime algorithm *SF*2, the group-based algorithm $SF4_g$, and the split algorithm *SF*5. In Table 13 we notice that there is no significant difference in the performances between these 3 algorithms since the calculated F value (2.16635) is smaller than the critical value (3.07376).

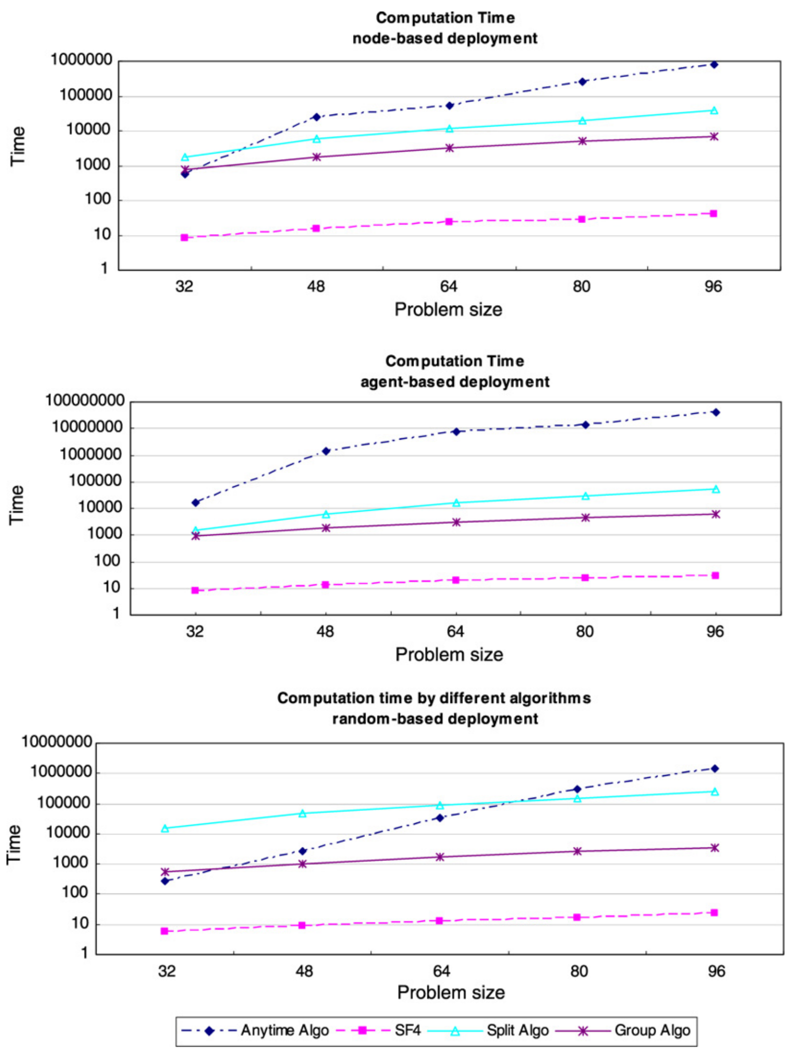


Fig. 2. Experiment 3: Computation time of the different algorithms with setting *s*3. The *x*-axis represents the problem size and the *y*-axis is the computation time (in a logarithmic scale).

Table 9
ANOVA on the results of 3 algorithms: *SF*2, *SF*5, and *SF*4_g in Experiment 3, with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	0.02316	1	0.02316	20.17751	2.412E–05	3.96347
Residual	0.08953	78	0.00115			
Total	0.11269	79				

Table 10
ANOVA on the results of 2 algorithms: *SF*2 and *SF*5 in Experiment 3, with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	0.00189	1	0.00189	2.45835	0.12095	3.96347
Residual	0.05998	78	0.00077			
Total	0.06187	79				

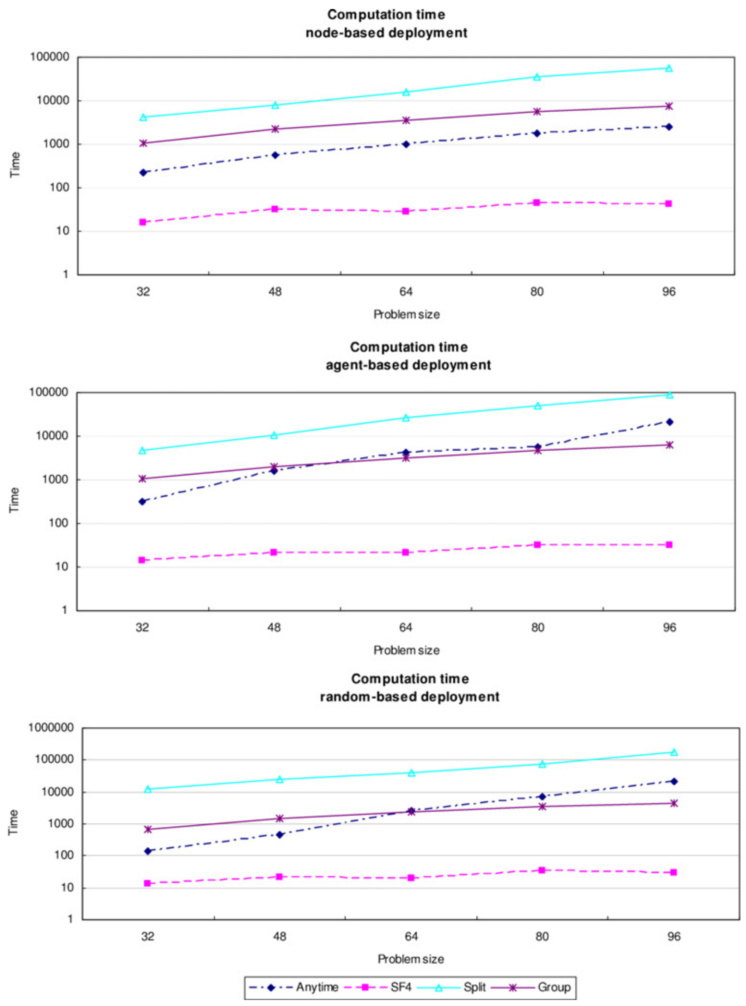


Fig. 3. Experiment 4: computation time of the different algorithms with setting $s4$. The x-axis represents the problem size and the y-axis is the computation time (in a logarithmic scale).

Table 11
Experiment 4: Upper bounds and approximation ratios of the different algorithms with setting $s4$.

Problem size	Deploy method	Upper bound	Anytime ($SF2$)	Disjoint ($SF4$)	Split ($SF5$)	Group-based ($SF4_g$)
$n20$, $a12$	node-based	0.972045	0.99988	0.999971	0.999996	0.999996
	agent-based	0.980811	0.999845	0.99981	0.99996	0.99994
	random-based	0.774173	0.991724	0.999837	0.999998	0.999999
$n30$, $a18$	node-based	0.973543	0.999922	0.999965	0.999983	0.999988
	agent-based	0.987256	0.99979	0.999535	0.999852	0.999823
	random-based	0.753271	0.991622	0.999918	0.999985	0.999983
$n40$, $a24$	node-based	0.979059	0.999723	0.999969	0.999985	0.999983
	agent-based	0.989146	0.999588	0.999269	0.999729	0.999681
	random-based	0.609391	0.991255	0.999639	0.999895	0.999889
$n50$, $a30$	node-based	0.971718	0.999661	0.999968	0.99989	0.999895
	agent-based	0.988569	0.999398	0.999136	0.999649	0.999638
	random-based	0.535532	0.990589	0.999655	0.999775	0.999787
$n60$, $a36$	node-based	0.978591	0.999681	0.999923	0.999957	0.999971
	agent-based	0.983952	0.999217	0.998105	0.999344	0.999097
	random-based	0.427743	0.976279	0.998948	0.999663	0.999713

Table 12
ANOVA on the results of 4 algorithms *SF*2, *SF*4, *SF*5, and $SF4_g$ in Experiment 4, with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	5.15015E–06	3	1.71672E–06	10.9514	1.4427E–06	2.6625
Residual	2.44542E–05	156	1.56758E–07			
Total	2.96043E–05	159				

Table 13
ANOVA on the results of 3 algorithms: *SF*2, *SF*5, and $SF4_g$ in Experiment 4, with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	4.09349E–07	2	2.04675E–07	2.16635	0.11917	3.07376
Residual	1.10541E–05	117	9.44791E–08			
Total	1.14634E–05	119				

Table 14
Experiment 5: Upper bounds and approximation ratios of the different algorithms in setting *s*1 but with a larger space ratio (3–4).

Problem size	Deploy method	Upper bound	Anytime (*SF*2)	Tree-based (*SF*3)	Disjoint (*SF*4)	Split (*SF*5)	Group-based ($SF4_g$)
$n18, a30$	node-based	0.973211	0.981061	0.98647	0.96855	0.998956	0.98707
	agent-based	0.976032	0.973769	0.986316	0.964855	0.999059	0.988083
	random-based	0.094253	–	–	–	–	–
$n24, a40$	node-based	0.980687	0.979052	0.991792	0.978524	0.99866	0.994861
	agent-based	0.980855	0.980291	0.992817	0.976784	0.998752	0.99174
	random-based	0.042501	–	–	–	–	–
$n30, a50$	node-based	0.983294	0.98326	0.975502	0.982723	0.998131	0.994358
	agent-based	0.982777	0.981894	0.975746	0.978989	0.997898	0.994035
	random-based	0.070558	–	–	–	–	–

Computation time. We show the computation times in Fig. 3, where unlike those shown in Experiment 3, the anytime algorithm *SF*2 in Experiment 4 converges pretty fast no matter what type of deployment employed. As explained above, in the resulting deployments in Experiment 4, the locations of many agents in the network are independent of one another. Consequently, we suppose that in Eq. (2), the importance of the sum of each term k ($k = 1, \ldots, |\mathcal{M}|$) for the value of Eq. (2) quickly decreases with the increase of k. Therefore, the anytime algorithm can converge faster in Experiment 4 than it did in Experiment 3. Again, the disjoint based algorithm *SF*4 was found to be the most efficient one.

Conclusion. We can conclude that for settings like *s*4, where there is relatively little overlap between the agents' locations in deployments, the disjoint-based algorithm *SF*4 is the best algorithm since it does very well on approximation and always returns solutions very fast.

Experiment 5

Experiment 5 repeats Experiment 1 with setting *s*1 but with an increased space ratio of nodes to agents from 2–3 to 3–4. We report the survivability results in Table 14. Compared with the results in Experiment 1 (see Table 1), the first noticeable difference in these results is in agent-based and in node-based deployments. In particular, there is a large increase in the upper bounds of the actual survivabilities from 0.3–0.44 (Table 1) to 0.97 (Table 14). Again, we did not include the survivabilities returned by the random based deployments in this table since they are relatively very low. Another noticeable change with the current setting is that the tree based algorithm *SF*3 is no longer the favorite—it returns lower survivabilities than the split and the group algorithms. Moreover, its accuracy is even lower than the disjoint based algorithm *SF*4 when the problem size is 80.

When more resources can be used to accommodate agents on the nodes, there are more valid future networks in the deployments compared to those in Experiment 1. Consequently, more valid future networks which have high survivabilities may not be included for the computation of *SF*3. Thus, *SF*3 in this set of experiments does not perform as well as it did in Experiment 1.

Statistical significance. Again we performed ANOVA on the results of each round of the different algorithms. Table 15 suggests that the differences of performances w.r.t. solution qualities of the different algorithms are significant since the F value (46.9782) is greater than the critical F value (2.41796).

Computation Time. Table 16 shows that *SF*3 is the most time-consuming algorithm with the problem size of 80 where it needs 100 times more computation time than the split and the group algorithms.

Table 15
ANOVA on the results of five algorithms in Experiment 5 with agent-based deployments and a problem size of 64. The level of significance required is set to 0.05.

	Sum of squares (SS)	Degrees of freedom (df)	Mean square (MS)	F value	p-value	F crit
Between algorithms	0.00325	4	0.00081	46.9782	1.303E–27	2.41796
Residual	0.00337	195	1.730E–05			
Total	0.00662	199				

Table 16
Experiment 5: Computation time (in microseconds) using different algorithms in setting $s1$ but with a larger space ratio (3–4).

Problem size	Deploy method	Anytime ($SF2$)	Tree-based ($SF3$)	Disjoint-based ($SF4$)	Split ($SF5$)	Group-based ($SF4_g$)
$n18, a30$	node-based	33674	19510	17	2372	1734
	agent-based	33253	19460	15	2224	1516
$n24, a40$	node-based	117114	146123	24	4844	2800
	agent-based	224809	148476	20	4576	2476
$n30, a50$	node-based	284858	838123	30	9177	4299
	agent-based	143220	839241	24	8744	3481

Table 17
Recommendations on the selection of the algorithms in different settings and with different criteria.

Setting	Resulting deployment	Solution requirement	Recommended algorithm
$s1$	many overlaps between agents' locations	quality and time	$SF3$
$s4$	few overlaps between agents' locations	quality and time	$SF4$
$s2$, $s3$, $s5$	many or few overlaps between agents' locations	quality and time	$SF5$
$s2$, $s3$, $s4$, $s5$	not many overlaps between agents' locations	time is critical, much more important than quality	$SF4$

Conclusion. Compared to the results of Experiment 1, in Experiment 5 the tree based algorithm $SF3$ is no longer the most favorite algorithm. Instead, the split algorithm $SF5$ outperforms others since it returns high quality solutions in relatively short computation time.

5.4. Summary and discussion

We provide recommendations for the choice of the algorithms in Table 17,[12] assuming either the node-based or the agent-based deployment methods are applied.

We show that in settings like $s1$, where node failures are distributed dramatically and the resources available in the network are very limited, the *tree based algorithm* $SF3$ performs extremely well.

However as in the case of setting $s4$ where the node failures are not distributed dramatically, and the deployments are those where most agents have disjoint locations w.r.t. other agents, the *disjoint based algorithm* $SF4$ is the best algorithm. Moreover, $SF4$ is recommended for applications where fast computation time is the most critical requirement, as long as the deployments will not result in too many overlaps between agents' locations (e.g. those generated in setting $s1$).

The *group based algorithm* $SF4_g$ is an improvement of $SF4$. As a result, $SF4_g$ returns a higher survivability than $SF4$ while it takes longer time. As in the case of $SF4$, this algorithm can be applied to environments where agents have disjoint locations on the nodes.

The *anytime algorithm* $SF2$ can be applied to applications which require flexible adjustments between the solution quality and the computation time—the solution returned by $SF2$ can be as accurate as possible as long as the time is affordable.

The *split algorithm* $SF5$ seems to be a general heuristic algorithm which provides good approximations. However when time is critical it may be preferable to use a faster heuristic algorithm like $SF4$.

[12] We made these recommendations based on the observations of the experimental results. It is possible that they may be inaccurate for some deployments.

6. Related work

We have introduced a probabilistic survivability model with various algorithms based upon the idea of replication. In this section, we first briefly review the work on fault tolerance (and particularly replication) in distributed systems and multi-agent survivability. We then discuss more replication based approaches in multi-agent systems.

6.1. Fault tolerance in distributed systems

Fault tolerance is the ability of a system to behave correctly even under the presence of faults. It aims to increase the *dependability* of a system, i.e., the ability of a system to perform the service that can be justifiably trusted [4] ([3] for detailed comparison). Fault tolerance has been extensively studied in distributed systems, and it is usually considered as a property of the system. Research in fault-tolerant distributed computing aims at making distributed systems more reliable by handling faults in complex computing environments [19]. In order to build a fault tolerant system, the first step is to specify the faults that the system may be subject to and thus must be tolerant to. Fault detection techniques identify the presence of an error. Fault handling methods are used for diagnosing faults and eliminating them from the system state. The choice of different techniques is strongly dependent upon the underlying fault assumption. We refer to [10,19] for detailed introductions on fault tolerance techniques in distributed systems.

An important characteristic which distinguishes multi-agent systems from traditional distributed systems is *autonomy*. Autonomy makes MASs more robust. Consequently, fault tolerance techniques that are designed for distributed systems may be difficult to directly apply to multi-agent systems. Furthermore, it is notoriously difficult to design fault tolerant systems [10], not to mention fault tolerant multi-agent systems due to their autonomy.

Replication is a well-known fault tolerance method for distributed systems. Wiesmann et al. [43] review several replication approaches in distributed systems according to (1) failure transparency for clients, and (2) server determinism. Services are implemented by multiple replicas on multiple servers. There are mainly three types of replication protocols: *active* replication, *passive* replication, and *semi-active* replication. The key concept of active replication is that all replicas receive and process every incoming request from a client concurrently. The replicas are deterministic. Therefore, failures are transparent to the clients, since if a replica fails, the others will still process the requests. In contrast to active replication, in passive replication, only one replica, called a primary replica, is contacted by the clients. The primary replica processes the requests from clients and then sends update messages to all other replicas. The passive replication is able to tolerate the non-deterministic servers. And it requires less computation resources than the active approach. However, it suffers from longer recovery delays when the primary replica fails. Semi-active replication does not involve the determinism problem because every time replicas need to make a non-deterministic decision, a leader replica makes the choice and sends it to the others. The selection of the replication protocol is dependent on the environment, such as the failure rate, and the application requirements. We refer to [20,43] for more details on replication in distributed systems.

The traditional replication based fault tolerant approaches in distributed systems usually define the replication protocols explicitly and statically at design time. Recently, dynamic data replication techniques have been investigated. Lin et al. [31] propose a centralized dynamic object replication algorithm which guarantees that at least t copies of the objects exist in a distributed system. Their goal is to minimize the total service cost of all the incoming requests. The t-availability is also guaranteed at any time instant as reported by Wolfson et al. in [44].

The focus of replication techniques in distributed computing is mainly on replication protocols and algorithms. Our work differs from theirs mainly because: (1) we do not develop the replication protocol but introduce a method to measure the quality of replications; and (2) we require and measure an *entire set* of agents to survive, rather than consider them individually as in the approaches of fault tolerant distributed systems.

6.2. Multi-agent survivability

The concept of *survivability* was introduced as a means of protecting critical systems. In earlier work, Ellison et al. [13] define survivability as *the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents*. Knight et al. [25,26] give a more precise definition of survivability based on specification: *a system is survivable if it complies with its survivability specification*. Their definition requires a complete, well designed survivability specification which contains six elements. However, there are some major open challenges in applying their definition to multiagent systems since it is very difficult to define some of the elements in their specification (e.g. such as enumerating all the states that the multiagent system might encounter in an open Internet style environment).

Survivability has been investigated in the context of multiagent systems recently by the UltraLog project [1,6,7,22]. UltraLog aims at ensuring the survivability of military logistics applications which are deployed on a large-scale distributed multi-agent system in dynamic and hostile environments. In their approach, a set of *measures of performance* have been used to determine the *overall success* of the system, which include, for instance, performance, availability, and integrity. They apply a hierarchy of control loops to guide survivability. Their efforts focus on architectural issues. As their survivability solutions are built on the Cougaar (Cognitive Agent Architecture) framework, in order to apply their approach to survivability, one must also develop the multi-agent applications on Cougaar and its UltraLog extensions and tools.

As an emerging discipline, survivability builds on related topics, such as fault tolerance, security, reliability, performance, verification and testing. There are multiple dimensions to the survivability of different systems and applications. Thus, the precise definition of the goals and the measurements of whether or not an application is a “success” or is “survivable” highly depends on the applications. Our approach to multi-agent survivability is distinguished from theirs as follows:

- We study survivability in the context of multi-agent systems. We propose a general and intuitive way to define the survivability of a multi-agent application.
- We developed several survivability algorithms that are able to provide precise measurements of how well multi-agent applications will survive under different environments.
- In contrast to the UltraLog approach where applicability is limited to a specific agent platform, our survivability model can be widely and easily applied to various multi-agent applications, independently of the agent’s development platform.

6.3. *Agent replication approaches*

Introducing redundant agents into a multi-agent system is an efficient way to improve the survivability of the MAS. Kumar et al. [29] propose an adaptive multi-brokered agent system, which applies replication to a *broker agent* (or middle agent). As the reduced number of brokers may degrade the system’s performance, they hypothesize that a broker team commits to maintain a *specified minimum number* of brokers. Thus, the system is robust to broker unavailability. However, the number of brokers in their system must be *predefined* by the broker teams without the guidance of any algorithms. In addition, they only consider the potential failures of brokers but ignore the possible faults of the regular agents.

Cloning and merging agents to support load balancing is discussed in [11,39]. Fan [14] furnishes each local agent with the capability of load-balancing. He proposes a BDI mechanism to formally model agent cloning for balancing agent workloads. These agent-cloning approaches mainly target the agent’s overload problem, while we aim to optimally deploy agents so that the survivability of the multi-agent system can be maximized.

In the context of mobile agent systems, survivability focuses on how to avoid the loss of agents during execution. Mishra and Huang [34] introduce a *Dependable Mobile Agent System* to recover from node and communication failures. Middle agents are distributed on every node in the network, which monitor the movement of agents and ensure that agents can arrive at their destinations reliably. Their approach deploys the agent replica on each node in the network. However, replication is expensive. Furthermore, they do not take into account the resource availability on the network. We propose various survivability algorithms to measure the quality of the deployments. Thus, it is possible to guide the agent replication based on probabilistic notions of node failures—something they do not consider.

Marin et al. [32,33] develop a *Dynamic Agent Replication* framework to design reliable distributed applications. Every agent in the MAS has a group of replicas, and a replication scheme is applied to each agent. At runtime, each agent can tune its internal parameters such as the number of its replicas. In their framework, the replication costs are assessed by simulations. Fedoruk and Deters [16,17] hide agent replication methods inside each agent. They propose a *transparent replication technique*, which makes the group proxy act as an interface between the replicas and the rest of the multi-agent system. In this manner, the proxies make the group appear to be a single entity and they control execution and state management of a replicate group. In [33] and [16,17], the importance of replicating agents optimally have been realized and they intend to minimize the additional complexity and system loads that are introduced by the use of replication. However, due to the lack of quality measurement of replication, in their approaches, it is the MAS designer’s responsibility to decide in advance for each agent, which, how many and where to deploy them.

The automatic and adaptive replication methods have been addressed in [5,8,21]. Briot et al. [8,21] propose a replication framework that allows adaptive control of the replication method, i.e. which and how many copies of agent to replicate, based on the *criticality* of each agent. In order to measure the criticality of each agent, the authors propose several strategies, such as an agent’s degree of dependence on other agents, the roles in the organization, its plan, etc. After estimating the criticality of all the agents, the number of replicas of each agent is then computed taking into account the criticality of agents, the minimum required replicas, and the current available resources. Similarly, Bora et al. [5] decide on the number of replicas in the system by measuring the agent’s importance. The feedback control theory is used to dynamically evaluate the criticality of agents, based on the information of failure rates and the agents’ roles. After receiving the criticality value from the feedback control mechanism, each agent in the organization calculates the desired number of replicas and then starts to clone or to kill its replicas subject to the available resources. The approaches of [5,8,21] identify the criticality of each single agent in the system in order to deploy the agent replicas. However, there is a lack of quality measurement of the resulting deployments as a team of agents. Our approach focuses more on the “application level”, i.e. given a task of MAS, we measure the survivability of this team of agents which work cooperatively to perform the task based on the information of the possible failures of the hosting nodes. It will be interesting to integrate the information of each agent’s criticality in a MAS into our approach of computing the survivability of the system.

In peer-to-peer applications, in order to maintain desired availability, Ranganathan et al. [37] deal with the issues of determining the number of replicas of any file and the location for new replicas. Similar to our model, they express the average probability of a node remaining operative, i.e. *stability*, which includes node failures, communication failures, and the disconnection of the node from the network. They compute the probability of replicas for each file being unavailable in the same manner as we calculate the single agent's survivability. Such computation problem is polynomial solvable. They do not study the availability or unavailability of the application which consists of multiple agents or files.

6.4. Discussion

In this paper, we have introduced an abstract probabilistic failure model, by using a disconnect probability function to represent the probability of the physical failure of a node which may result in the malfunction of its deployed multi-agent application. Our assumptions for this failure model are (1) the independence of failures between nodes, and (2) the fully connected network. We do not make any further assumptions, such as cause of failures, failure detection method, failure rate, and calculation of the disconnection probability. These, however, can be detailed in the failure model when it is applied to specific applications. For instance, the disconnect probability of a node may be estimated from the statistical data, or be measured by round trip time between this node and others. It may also take into account time in the probabilistic failure model, as that of [23]. We have shown in Section 2.1 the possible ways to specify the disconnection probability in the context of the CoAX application, the Skoda application, and Tichy's ship control application. Furthermore, the second assumption about fully connected networks assumed in this paper can be relaxed in some way. For example, given a partially connected agent system, the disconnection of a node can be considered as the event that this node becomes *unreachable* in the system. In this way, the model can be used to represent different environments where the MAS application is deployed and works.

Our goal in this paper was not to develop a replication scheme for multi-agent systems. Instead, we assume that the deployment (of replicas) is already given, and we introduce various algorithms to measure the survival level of the deployment. Hence, our approach does not make any hypotheses about the replication protocol and its implementation, such as active or passive replication, deployment methods, etc. However, we could select and use the existing replication protocols or deployment methods described in related work for specific applications. For instance, for disaster management or military applications where a fast recovery delay is critical, the active replication protocol can be chosen, and a fast survivability algorithm can be used to estimate whether the current deployment meets the minimal survival requirement. Another example is a multi-agent application consisting of highly heterogeneous agents, for which we can apply the methods in [5,8] to deploy replicas subject to the importance of agents. By selecting a proper survivability algorithm, together with these deployment methods, we are able to find a high quality multi-agent deployment. Therefore, the contribution of our work is the ability to *guide* the replication by assessing the resulting deployments associated with the current condition of the failure model.

Likewise, there are many different ways of maintaining consistency amongst multiple replicas of a piece of data or software. In databases, there is a long history of methods [43,44] used to maintain consistency amongst multiple replicas. These are done primarily through the use of *checkpointing* methods—timestamps adorn changes to the data and consistency is maintained by using these timestamps to update replicas to maintain consistency. Many papers on MASs distinguish between a state that the agent has at a given point in time and the behavior of the agent that is often encoded via certain types of rules [40]. In such cases, the state can be stored in a relational database. The rules do not change as the agent operates and hence, we only need to ensure that the states of different agents are synchronized. The numerous techniques to synchronize replicated relational databases can now be applied here.

Another relevant problem that has been extensively studied is that of understanding when nodes in a network go down [15,28]. Our framework can be used in conjunction with *any* existing method to determine when a network node goes down.

7. Conclusion

Kraus et al. [27] were the first to propose a probabilistic notion of fault tolerance of a multi-agent system based upon replication principles. Their algorithms for solving the most survivable deployment problem (finding a deployment that has the highest survival probability) include two elements:

(1) An algorithm to solve the deployment survivability problem (measuring the survival probability of a given deployment) under the *assumption that we are ignorant about the relationships between node failures in a network* and
(2) An algorithm that uses the previous algorithm to find the deployment that has the highest probability of survival.

In this paper, we have focused on the deployment survivability problem, and we have studied this problem under the assumption of independence of node failures. We have made the following contributions.

(1) We have proven that the deployment survivability problem is at least NP-hard (under the independence assumption) and hard to approximate up to a factor of $2^{|V|^{1-\epsilon}}$.
(2) We have proven that the most survivable deployment problem is at least NP-hard (under the independence assumption). Moreover, we have proven that *any* polynomial-time approximation algorithm is bound to provide maximally bad answers in some cases unless $\mathsf{P} = \mathsf{NP}$.

(3) We then presented two algorithms to accurately compute the probability of survival of a given deployment. One of these algorithms is exponential in the number of agents, while the other is exponential in the number of nodes in the network. Thus, if one of these quantities is small, we can use this algorithm to accurately compute the survival probability of a deployment.
(4) We then presented a set of five different heuristic algorithms to compute the survival probability of a deployment.
(5) Finally we have compared the performance of our algorithms according to the quality of the solution found (i.e. how close the solution found by the heuristic is to either the correct solution or to a bound on the solution in cases where the solution cannot be accurately computed) and in terms of computation time. In all cases, we tried to use statistical significance tests to determine if our inferences had statistical significance. We did this under five different environmental settings. Our results show that the performance of most of the algorithms, namely the tree-based algorithm *SF*3, the disjoint based algorithm *SF*4, the group based algorithm $SF4_g$, and the anytime algorithm *SF*2, vary greatly with the environment settings—that is, each of these algorithms is appropriate for certain settings, but not for others. In contrast, the split algorithm *SF*5 has demonstrated a relatively stable performance in terms of quality. Nonetheless its running time is dependent on the setting and the problem size.

In addition, we believe some existing MAS replication frameworks may benefit from the proposed algorithms in order to compute the survival level of the resulting deployments, and thus result in better fault tolerant multi-agent systems. Integrating the proposed method into existing replication frameworks would be interesting to address in future work.

One problem with the approach described in this paper is that it is *static* in the sense that it does not *adapt* to changes that affect the survivability of the MAS. However, it provides a basis for the development of an adaptive approach. An adaptive approach cannot be built without learning how to compute the survival probability of a candidate deployment. For instance, if there is any change or failure in the network, the algorithms proposed in this paper could be used to first quickly check if the survivability of the current deployment is too low (or invalid). If this is the case, the algorithms then calculate a better new deployment according to the new environment parameters, and agents could be re-deployed to new locations accordingly. In this manner, we would be able to ensure maximal survivability of an agent application in dynamic, changing environments.

A second problem of the developed survivability algorithms is that they are *centralized*. So even though the agents are distributed across the network, the survivability algorithm itself resides on a single node. One solution to this problem could be distributed algorithms which are built on top of the centralized survivability algorithm. We are currently working on such an extended distributed, adaptive approach, based on our previous work [41].

A third major topic for future work would be to use a mixture of assumptions when computing the probability of survival of a MAS. As this paper shows through the CoAX example, the Skoda example, and the ship onboard control example, in many applications, there is a mixture of assumptions about node failures that can be used. For example, in the CoAX example, node failures in the UK and US may be independent of node failures of sensor nodes. However, there may be dependencies between failures of sensor nodes. One way to do this would be to adapt probabilistic conjunction strategies (PCSs) proposed in [30] as an extension of the notion of a triangular norm [24]. PCSs are functions satisfying certain axioms that provide methods to compute the tightest probability interval of an event $(e_1 \wedge e_2)$ given a probability interval for each of e_1, e_2. Lakshmanan et al. [30] propose a set of axioms that PCSs must obey. They show that the ignorance (of node failures) assumption used in [27] and the independence assumption used in this paper are both special cases of PCSs. Future work could examine how to replace disconnect probability functions proposed in this paper with an extended disconnect probability function *epd* that expresses statements such as $edp(n_1) = 0.3$ denoting that n_1's disconnect probability is 0.3, $epd(n_1 \wedge n_2) = epd(n_1) \otimes epd(n_2)$ where $\otimes$ is a conjunction strategy. Thus, the syntax used to represent *epd*'s would allow joint probabilities to be specified. A *disconnect probability specification* would then be a set of equations of the form mentioned above. A major challenge would be to extend the methods and results of this paper when a disconnect probability specification of this type is used.

There are many other interesting directions for future work that are related to this topic. In many real-world applications, it is essential to maintain a *minimal level of survival*. Thus, one variant of the current approach would be to develop algorithms for deployments that meet such minimal survival requirements. Another topic would be to study the survivability of a dynamically changing multi-agent system, instead of a fixed one. This could be useful for multi-agent applications where agents are connected intermittently. Another issue is the trade-off between the survivability of a multi-agent system and its performance. Ensuring survivability could be costly due to, for example, state synchronization among replicas and computing survivability. Consequently this comes at the cost of actually providing the services the multi-agent system is supposed to provide. Such problems are significant for multi-agent applications with scarce resources. Thus, it would be worthwhile to study how these two concerns could be balanced.

Yet another important issue is "gaming" the system. For example, suppose the methods described in this paper are used to implement MAS security for an application. A user who knows that the techniques of this paper are used in the system can try to utilize this knowledge in order to break security. This leads to a game theoretic framework whereby we need to reason about how an adversary would make use of such knowledge in order to break the system. The root node of the game tree consists of the state of the system. The children of the node refer to the possible states resulting from an action that an adversary could take. The system then needs to make a "move" in order to determine how best to respond to the user's action. This is an important area which we plan to study in the future.

References

[1] Ultralog program site, http://www.ultralog.net.

[2] D. Allsop, P. Beautement, M. Kirton, J. Bradshaw, N. Suri, E. Durfee, C. Knoblock, A. Tate, C.W. Thompson, Coalitions agent experiment: Multiagent cooperation in International Coalitions, IEEE Intelligent Systems 17 (3) (2002) 26–35.

[3] A. Avizienis, J. Laprie, B. Randell, Fundamental concepts of dependability, in: Proceedings of the Third International Workshop on Information Security, ISW, 2000.

[4] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 01 (1) (2004) 11–33.

[5] S. Bora, O. Dikenelli, Applying feedback control in adaptive replication mechanisms in fault tolerant multi-agent organizations, in: SELMAS '06: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, New York, NY, USA, ACM, 2006, pp. 5–12.

[6] M. Brinn, J. Berliner, A. Helsinger, T. Wright, M. Dyson, S. Rho, D. Wells, Extending the limits of DMAS survivability: The ultralog project, IEEE Intelligent Systems 19 (5) (2004) 53–61.

[7] M. Brinn, M. Greaves, Leveraging agent properties to assure survivability of distributed multi-agent systems, in: AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, ACM, 2003, pp. 946–947.

[8] J.-P. Briot, Z. Guessoum, S. Aknine, A.L. Almeida, J. Malenfant, O. Marin, P. Sens, N. Faci, M. Gatti, C. Lucena, Experience and prospects for various control strategies for self-replicating multi-agent systems, in: SEAMS '06: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems, New York, NY, USA, ACM, 2006, pp. 37–43.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, New York, 1990.

[10] F. Cristian, Understanding fault-tolerant distributed systems, Communications of the ACM 34 (2) (1991) 56–78.

[11] K. Decker, K.P. Sycara, M. Williamson, Cloning in intelligent, adaptive information agents, in: C. Zhang, D. Lukose (Eds.), Multi-Agent Systems: Methodologies and Applications, Springer-Verlag, 1997, pp. 63–75.

[12] J.L. Devore, Probability and Statistics for Engineering and the Sciences, 4th edition, Wadsworth Publishing, 1995.

[13] R.J. Ellison, D.A. Fisher, R.C. Linger, H.F. Lipson, T.A. Longstaff, N.R. Mead, Survivability: Protecting your critical systems, IEEE Internet Computing 3 (6) (1999) 55–63.

[14] X. Fan, On splitting and cloning agents, Technical Report 407, Turku Center for Computer Science, Turku, Finland, 2001.

[15] F. Feather, D. Siewiorek, R. Maxion, Fault detection in an Ethernet network using anomaly signature matching, in: Proceedings of the International Conference on Communications architectures, protocols and applications, September 13–17, 1993, San Francisco, CA, USA, pp. 279-288.

[16] A. Fedoruk, R. Deters, Improving fault-tolerance by replicating agents, in: C. Castelfranchi, W.L. Johnson (Eds.), Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, ACM Press, 2002, pp. 737–744.

[17] A. Fedoruk, R. Deters, Using dynamic proxy agent replicate groups to improve fault-tolerance in multi-agent systems, in: AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, ACM, 2003, pp. 990–991.

[18] J. Galambos, I. Simonelli, Bonferroni-type Inequalities with Applications. Probability and its Applications, Springer-Verlag, 1996.

[19] F.C. Gartner, Fundamentals of fault-tolerant distributed computing in asynchronous environments, ACM Computing Surveys 31 (1) (1999) 1–26.

[20] R. Guerraoui, A. Schiper, Software-based replication for fault tolerance, IEEE Computer 30 (4) (April 1997) 68–74.

[21] Z. Guessoum, N. Faci, J.-P. Briot, Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform, SIGSOFT Softw. Eng. Notes 30 (4) (2005) 1–6.

[22] A. Helsinger, K. Kleinmann, M. Brinn, Framework to control emergent survivability of multi-agent systems, in: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society, 2004, pp. 28–35.

[23] F. Klein, M. Tichy, Building reliable systems based on self-organizing multi-agent systems, in: SELMAS '06: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Shanghai, China, ACM, 2006, pp. 51–58.

[24] E.P. Klement, R. Mesiar, E. Pap, Triangular Norms, Kluwer Academic Publishers, ISBN 0-7923-6416-3, 2000.

[25] J.C. Knight, E.A. Strunk, Achieving critical system survivability through software architectures, in: Architecting Dependable Systems II, in: Lecture Notes in Computer Science, vol. 3069, Springer, 2004, pp. 51–78.

[26] J.C. Knight, K.J. Sullivan, M.C. Elder, C. Wang, Survivability architectures: Issues and approaches, in: DARPA Information Survivability Conference and Exposition, Los Alamitos, CA, IEEE Computer Society Press, 2000, pp. 157–171.

[27] S. Kraus, V.S. Subrahmanian, N.C. Tacs, Probabilistically survivable MASs, in: G. Gottlob, T. Walsh (Eds.), IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003, Morgan Kaufmann, 2003, pp. 789–795.

[28] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen, Sketch based change detection: Methods, evaluation and application, in: Proc. 3rd ACM SIGCOMM Conference on Internet Measurement. Miami Beach, FL, 2003, pp. 233–247.

[29] S. Kumar, P.R. Cohen, H.J. Levesque, The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams, in: E. Durfee (Ed.), The Fourth International Conference on Multi-Agent Systems (ICMAS 2000), IEEE Press, 2000, pp. 159–166.

[30] V.S. Lakshmanan, N. Leone, R. Ross, V.S. Subrahmanian, Probview: A flexible probabilistic database system, ACM Transactions on Database Systems 22 (3) (1997) 419–469.

[31] W.J. Lin, B. Veeravalli, A dynamic object allocation and replication algorithm for distributed systems with centralized control, Int. J. Comput. Appl. 28 (1) (2006) 26–34.

[32] O. Marin, M. Bertier, P. Sens, DARX—a framework for the fault-tolerant support of agent software, in: ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering, Washington, DC, USA, IEEE Computer Society, 2003, pp. 406–418.

[33] O. Marin, P. Sens, J.-P. Briot, Z. Guessoum, Towards adaptive fault tolerance for distributed multi-agent systems, in: Proceedings of the 3rd European Research Seminar on Advanced Distributed Systems (ERSADS'2001), 2001, pp. 195–201.

[34] S. Mishra, Y. Huang, Fault tolerance in agent-based computing systems, in: Proceedings of the 13th ISCA International Conference on Parallel and Distributed Computing Systems, 2000, pp. 413–426.

[35] R. Mittu, R. Ross, Building upon the coalitions agent experiment (coax)—integration of multimedia information in gccs-m using impact, in: Proceedings of the 9th International Workshop on Multimedia Information Systems Ischia, Italy, pp. 35–44.

[36] M. Pechoucek, V. Marik, Review of industrial deployment of multi-agent systems, Technical Report, Gerstner Laboratory, Agent Technology Group, Department of Cybernetics, Czech Technical University in Prague, Czech Republic and Rockwell Automation Research Center, Prague, Czech Republic, http://agents.felk.cvut.cz/teaching/33ui2/on-applications.pdf.

[37] K. Ranganathan, A. Iamnitchi, I. Foster, Improving data availability through dynamic model-driven replication in large peer-to-peer communities, in: CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Washington, DC, USA, IEEE Computer Society, 2002, p. 376.

[38] D. Roth, On the hardness of approximate reasoning, Artificial Intelligence 82 (1–2) (1996) 273–302.

[39] O. Shehory, K.P. Sycara, P. Chalasani, S. Jha, Increasing resource utilization and task performance by agent cloning, in: ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, London, UK, Springer-Verlag, 1999, pp. 413–426.

[40] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, R. Ross, Heterogeneous Agent Systems, MIT Press, Cambridge, MA, 2000.

[41] V.S. Subrahmanian, S. Kraus, Y. Zhang, Distributed algorithms for dynamic survivability of multiagent systems, in: Proc. Computational Logic in Multi-Agent Systems: 4th International Workshop, CLIMA IV, in: Lecture Notes in Computer Science, vol. 3259, Springer-Verlag, 2004, pp. 1–15.
[42] P. Tichy, P. Slechta, F. Maturana, S. Balasubramanian, Industrial MAS for Planning and Control, Lecture Notes in Computer Science, vol. 2322, Springer-Verlag.
[43] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, Understanding replication in databases and distributed systems, in: T.-H. Lai (Ed.), Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000), IEEE Computer Society Technical Committee on Distributed Processing, 2000, pp. 264–274.
[44] O. Wolfson, S. Jajodia, An algorithm for dynamic data distribution, in: Workshop on the Management of Replicated Data, 1992, pp. 62–65.
[45] Y. Zhang, E. Manister, S. Kraus, V.S. Subrahmanian, Approximation results for probabilistic survivability, in: Second IEEE Symposium on Multi-Agent Security and Survivability, Philadelphia, USA, 2005, pp. 1–10.