Expert Systems With Applications 211 (2023) 118627

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Online distributed evolutionary optimization of Time Division Multiple Access protocols

Anil Yaman [a,*], Tim van der Lee [b], Giovanni Iacca [c]

[a] Department of Computer Science Vrije, Universiteit Amsterdam, Amsterdam, 1081 HV, The Netherlands
[b] Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, 5600MB, The Netherlands
[c] Department of Information Engineering and Computer Science, University of Trento, Trento, 38123, Italy

ARTICLE INFO

Keywords:
Distributed evolutionary algorithm
Network protocol
Online adaptation
Time Division Multiple Access
Multi-objective optimization

ABSTRACT

With the advent of cheap, miniaturized electronics, ubiquitous networking has reached an unprecedented level of complexity, scale and heterogeneity, becoming the core of several modern applications such as smart industry, smart buildings and smart cities. A crucial element for network performance is the protocol stack, namely the sets of rules and data formats that determine how the nodes in the network exchange information. A great effort has been put to devise formal techniques to synthesize (offline) network protocols, starting from system specifications and strict assumptions on the network environment. However, offline design can be hard to apply in the most modern network applications, either due to numerical complexity, or to the fact that the environment might be unknown and the specifications might not available. In these cases, online protocol design and adaptation has the potential to offer a much more scalable and robust solution. Nevertheless, so far only a few attempts have been done towards online automatic protocol design. These approaches, however, typically require a central coordinator, or need to build and update a model of the environment, which adds complexity. Here, instead, we envision a protocol as an emergent property of a network, obtained by an environment-driven Distributed Hill Climbing (DHC) algorithm that uses node-local reinforcement signals to evolve, at runtime and without any central coordination, a network protocol from scratch, without needing a model of the environment. We test this approach with a 3-state Time Division Multiple Access (TDMA) Medium Access Control (MAC) protocol and we observe its emergence in networks of various scales and with various settings. We also show how DHC can reach different trade-offs in terms of energy consumption and protocol performance.

1. Introduction

A fundamental element in many engineering and industrial applications is the use of networked systems: be it environment monitoring, smart industries, smart cities, or distribution systems, networks of various scales and complexity are employed today practically everywhere. One of the most important aspects in network design is the protocol stack, which determines the way (data format and rules) the nodes in a network communicate with each other (Holzmann, 1991).

Traditionally, network protocols have been modeled as a *reactive system*, i.e., a two-player game where an agent (a node in the network) *reacts* – by performing a certain action – to predefined conditions in the environment (the rest of the network): for instance, the agent retries a packet transmission if it does not receive an acknowledgment. As such, a protocol can be described with an automaton, for which formal specifications can be logically expressed and verified. For that, one usually needs to have complete knowledge about (and strict assumptions on) the environment. This approach, rooted in the theory of Temporal Logic and infinite Büchi automata (Büchi & Landweber, 1990), has been the gold standard in protocol design and verification for decades. Since the late '60, an impressive number of theoretical and practical results have been obtained in this area, gearing towards the automatic synthesis of protocol from service specifications (Saleh, 1996) and the development of automatic model checker tools, such as SPIN (Holzmann, 1997).

Despite these many successes, this approach to protocol design has also limitations. First of all, it assumes, in general, the environment – and all its states – to be known: this might not be the case of some modern network applications, where the environment conditions might be unpredictable. Furthermore, this approach models the environment *as a whole*, i.e., without describing the mutual interactions between the other nodes in the network. Two notable exceptions to this approach, that instead model also these interactions, are the reported in Al Dallal and Saleh (2012) and Finkbeiner and Gölz (2017). Another issue is the

* Corresponding author.
E-mail addresses: a.yaman@vu.nl (A. Yaman), t.lee@tue.nl (T. van der Lee), giovanni.iacca@unitn.it (G. Iacca).

https://doi.org/10.1016/j.eswa.2022.118627
Received 27 April 2022; Received in revised form 16 August 2022; Accepted 16 August 2022
Available online 27 August 2022

numerical (time and space) complexity of these methods, which makes them impractical when the number of protocol and environment states grows (Vardi, 2018). Finally, and this is arguably the main limitation, this approach fundamentally follows a *waterfall design model*, as the design and verification steps are performed *offline*, before deployment, and usually no online adaptation or feedback design cycles are considered. This is the case of projects such as x-kernel (Hutchinson & Peterson, 1991), Horus and Ensemble (Birman et al., 2000) and—to some extent—ANTS (Wetherall et al., 1998): all these tools offer a great level of modularity and abstraction, but—as pointed out in Keller et al. (2008)—their being based on offline design limits their flexibility and applicability. With the exception of ANTS, which also provides a way to dynamic reprogramming/redeploying the protocol code over the network, in all the other cases if one wants to change the protocol code all network nodes must be manually reprogrammed with the new protocol implementation: clearly, not only this approach disrupts the operation of the network, but also it becomes expensive when the network size increases.

In contrast to this traditional "rigid" offline approach, online protocol design and adaptation (Lee et al., 2020) has the potential to offer a much more scalable and robust solution. Some researchers have even suggested that flexibility – in the form of self-adaptation and *empowerment*, i.e., the principle of agents performing the actions which maximize the number of reachable states – should be the key design principle in modern network engineering (Kalmbach et al., 2018; Kellerer et al., 2015, 2019). Nevertheless, so far only a few attempts have been done in this direction, oriented towards the grand vision of *autonomic* or *self-adaptive* networking (Bouabene et al., 2009; Xiao, 2016). Most of these attempts are based on Machine Learning (ML) and bio-inspired techniques, although they suffer from some limitations, as we briefly summarize below.

Machine Learning: Various solutions based on collective intelligence (Wolpert et al., 1999) and Reinforcement Learning (RL) (Peshkin & Savova, 2002; Stampa et al., 2017; Tao et al., 2001) have been proposed in the context of routing protocols and particularly in IoT systems (Woźniak et al., 2020), such as Wireless Sensor Networks (WSNs) (Alsheikh et al., 2014; Förster & Murphy, 2011; Kulkarni et al., 2010); other works (He et al., 2019) used Deep Learning (DL) to model and optimize the physical layer, or optimization and consensus mechanisms to reduce energy consumption (Wadhwa et al., 2022; Woźniak et al., 2021). Albeit quite powerful, the main limitation of most of these approaches is that they often require a large amount of data collected from the network at runtime, in order to build (i.e., train offline) a model of the protocol, to be used later for online adaptation and optimization. In this sense, we can consider these methods as "semi-online".

Bio-inspired techniques: a large body of research exists in this field, as surveyed for instance in Nakano (2010). Similarly to the ML-based methods, most of the existing literature focuses however on offline optimization, based on Swarm Intelligence algorithms, such as Particle Swarm Optimization (Guo & Lv, 2020) or Ant Routing (Zhang et al., 2020), and especially on Evolutionary Algorithms (EAs)[1]. In the context of EAs, a seminal paper is represented by the study (Elfakih et al., 1999). Later on, Genetic Programming (GP) has been successfully used to *evolve* – i.e., optimize offline – protocol adaptors (Van Belle et al., 2003), wireless protocols based on Carrier Sense Multiple Access (CSMA) (Tekken-Valapil & Kulkarni, 2017), aggregation protocols (Weise et al., 2007; Weise & Tang, 2011; Weise et al., 2008), or MAC access protocols (Lewis et al., 2006; Roohitavaf et al., 2018). The latter have been evolved also by means of evolvable Finite State Machines (FSMs) (Hajiaghajani & Biswas, 2015a, 2015b; Sharples & Wakeman, 2000). As for online methods, an interesting bio-inspired (distributed) learning approach was introduced in Su and Van Der Schaar (2010), where each node observes the other nodes' behavior and forms internal conjectures on how they would react to its actions, to then choose the action that maximizes a local utility function: the authors demonstrated, analytically and through numerical simulations, that this method reaches Nash equilibria corresponding to optimal traffic fairness and throughput. Other works have investigated distributed EAs (Iacca, 2013) and distributed GP (Johnson et al., 2005; Valencia et al., 2010) to evolve the nodes' parameters and functioning logics of WSNs, or distributed optimization in multi-agent network systems. Finally, two notable online methods are STEM-Net (Aloi et al., 2014) and Fraglets (Yamamoto et al., 2007). The first one is a wireless network where each node uses an EA "to reconfigure itself at multiple layers of the protocol stack, depending on environmental conditions, on the required service and on the interaction with other analogous device" (Aloi et al., 2014). The latter is based on the concept of "autocatalytic software" (Tschudin & Yamamoto, 2005), or chemical computing (Miorandi & Yamamoto, 2008): essentially, protocols emerge automatically as collections of "fraglets", i.e., combinations of code segments and parameters which are evolved, respectively, by distributed GP (Yamamoto & Tschudin, 2005) and distributed EAs (Alouf et al., 2010), and spread over the network through opportunistic (epidemic) propagation (Alouf et al., 2007) regulated by interactions with the environment. On top of this, an additional EA optimizes the combination of protocols (Imai & Tschudin, 2010).

In this work, we continue along this research direction on online evolution of protocols. We consider a network of spatially distributed, locally connected nodes, and – to illustrate our method's applicability – we focus on a Time Division Multiple Access (TDMA)-based Medium Access Control (MAC) protocol, where given a time frame consisting of a fixed number of time slots, each node should learn in which slots it should transmit, listen, or stay idle, see Fig. 1 for an example. TDMA protocols are of particular interest in modern network research since the existing protocols are usually not suitable to handle the most recent network instances such as WSNs (Lee & Cho, 2017), vehicular ad-hoc networks (Sun et al., 2020) or underwater acoustic networks (Yun et al., 2013). On the other hand, this approach is applicable also to other protocols and stack layers.

Our proposed solution works as follows. Starting from a *tabula rasa*, nodes automatically learn the optimal protocol configuration – the one that minimizes collisions and/or reduces the energy consumption (for which we use, as proxy, the number of slots used for transmit/listen actions) – online, and in a distributed manner. To do this, we propose an environment-driven approach, based on our previous work (Yaman & Iacca, 2021), in which each node in the network runs *in situ* a minimalist evolutionary search, receives reinforcement signals corresponding to its actions, and occasionally shares its protocol parameters with its neighbors. This approach aims to optimize the collective behavior of a population of agents that interact with a certain environment, and uses such interaction to drive a distributed evolutionary search.

We should note that similar approaches have has been proposed in earlier literature, although those were primarily focused on "embodied" collective robotics (Bredeche et al., 2018). This concept was initially conceptualized, for the case of a single robot, in Eiben et al. (2010), and later extended to the case of distributed evolution (over groups of robots) in Bredeche et al. (2012), including environment-driven adaptation (Haasdijk et al., 2014), and real-time constraints (Prieto et al., 2016). However, to the best of our knowledge an approach of this kind has never been applied to the evolution of network protocols.

Compared to traditional (offline) protocol design, as well as to other methods for MAC protocol optimization, the advantages of our approach are manifold:

[1] Interestingly, a loop between network engineering and evolutionary theory exists. Recent evidence has shown that the "hourglass" shape of most protocol stacks is the result of an implicit evolutionary process that led to a minimal complexity, maximal robustness architecture (Akhshabi & Dovrolis, 2011; Dovrolis, 2008; Siyari et al., 2017).

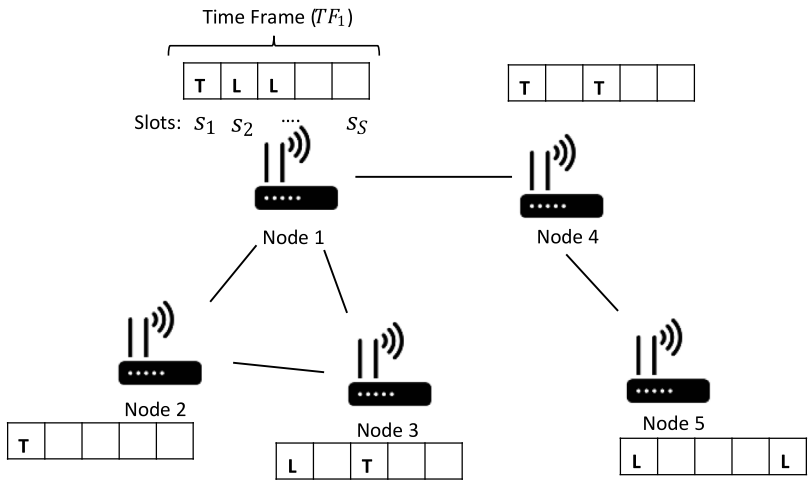


Fig. 1. Example network model used for evolving TDMA MAC protocols: each node has a time frame consisting of S slots, where each slot $s_i, i \in \{1, \dots, S\}$ can be either "T" (transmit), "L" (listen) or empty (idle).

1. **Scalability**: since there is no central unit that guides the evolutionary search, this method can potentially scale up to very large networks.
2. **Robustness**: since the evolutionary search runs continuously and open-ended, it allows a form of continual learning that can respond adequately to different network conditions, even unknown prior to deployment.
3. **Environment-independence**: the proposed method is fairly agnostic to the environment: by definition, the other nodes' behavior – in our case, the structure of their time frames – is not known a priori as it will adapt online via embedded evolution. Furthermore, differently from other works such as Su and Van Der Schaar (2010), there is not even the need to build a node-local conjecture on how the other nodes would behave. In essence, our only assumptions are: (1) the node actions are taken from a finite action set that is the same for all nodes in the network (in our case: transmit, listen, idle); (2) each action is associated to a certain reinforcement signal that is the same in all the network; (3) for each transmission an ack packet can be received at the transmitter to acknowledge that the transmitted packet has been received correctly.
4. **Full decentralization**: one distinct characteristic of our proposal w.r.t. the existing evolutionary approaches to MAC protocol optimization, which are typically centralized, such as the aforementioned works based on GP (Lewis et al., 2006; Roohitavaf et al., 2018) or those based on evolved FSMs (Hajiaghajani & Biswas, 2015a, 2015b; Sharples & Wakeman, 2000), is that our approach is fully distributed (decentralized) and works online rather than offline.

The remaining of this paper is organized as follows. In the next section, we introduce the TDMA problem and the proposed Distributed Hill Climbing algorithm. In Section 3, we present the experimental setup, while the numerical results are discussed in Section 4. Finally, in Section 5 we draw the conclusions and hint at future works.

2. Methods

In this section, we present first (Section 2.1) the problem settings, i.e., how we model the network and how we perform the evaluation process. Then, we describe the proposed method (Section 2.2).

2.1. Problem settings

The network, illustrated in Fig. 1, can be represented as a graph $G = (V, E)$ consisting of a set of nodes V, and undirected edges $E \subset V \times V$ which represent the possibility of communication between the nodes (Ergen & Varaiya, 2010). Two nodes, seed and target, are selected in the network. All the packets originating from the seed node and are targeted to the target node.

Each node n_i has a time frame, $\boldsymbol{TF}_i$, consisting of S time slots that can take one of three values: "T", "L" or empty, to indicate transmit, listen, and idle actions, respectively. We assume that the time frames of all nodes have the same length and that their time slots are synchronized. In addition, each node has a queue data structure Q_i of unlimited size to allow storing packets if they cannot be transmitted.

The evaluation process of the whole network is performed for K time steps. The process starts at time step $t_1 = 1$ by simultaneously executing the action in the first time slot of each node. Then, at each time step $t_k, k \in \{2, \dots, K\}$, the action to be executed is the one in the next slot in the time frame, until the last slot is reached. Once the last slot is reached, the next action to be executed becomes the one from the time slot, and so on. This process is repeated until the last time step t_K is reached.

During the evaluation process, a predefined number of packets are introduced into the queue of the seed node with a certain frequency. Depending on the actions in the time slots of the nodes, these packets are transmitted and received across nodes. For instance, if there is a packet in the queue of a node, and the action in the current time slot is "T", then the node attempts to transmit the packet. Every node in the neighborhood that performs the action "L" in the same time step can receive the packet. However, a *collision* occurs when a node performing the action "L" has more than one neighbor that is attempting to transmit a packet at the same time. This can be observed in Fig. 1. In the first time slot, s_0, while "Node 3" listens, its two neighbors "Node 1" and "Node 2" try to transmit at the same time, which causes a collision. In this case the packet cannot be received. If there are no collisions, the packet is received by the neighbor and put into its queue. The neighbor that receives the packet sends an acknowledgment. In this case, the transmitting node removes the packet from its queue. If a packet is received at the target node, it is removed from its queue. The number of packets received at the target node and the number of time steps took for each packet to reach the target are recorded.

The goal of the problem is to assign an action to each time slot of the time frame of each node in such a way to allow all packets generated at the seed node to reach the target node (100% delivery rate). Ideally, it is also desirable to reduce the number of used slots (i.e., slots assigned to transmit and listen actions), to reduce the network energy consumption.

As a note, to avoid multiple transmissions of the same packet we include an additional data structure to keep track of the received

Table 1
Node behavior reinforcement associations (NBRAs).

Behavior ID	Action	Queue	Outcome	r_k
1	Transmit	Empty	–	r_1
2	Transmit	Non-empty	(ack) received	r_2
3	Transmit	Non-empty	(ack) not received	r_3
4	Idle	Empty		r_4
5	Idle	Non-empty		r_5
6	Listen	Empty	(packet) received	r_6
7	Listen	Non-empty	(packet) received	r_7
8	Listen	Empty	(packet) not received	r_8
9	Listen	Non-empty	(packet) not received	r_9

packets in each node. To achieve that, we simply record the packet ID of each packet in each node when they are received. Moreover, if a node receives a packet with the same ID that was received before, it simply ignores it and does not send an acknowledgment.

2.2. Proposed method: Distributed Hill Climbing

Algorithm 1 shows the proposed Distributed Hill Climbing (DHC) algorithm for optimizing TDMA MAC protocols over a network consisting of N nodes. Each node $n_i, i \in \{1, \dots, N\}$ runs the DHC algorithm independently. The DHC algorithm.[2] starts with the initialization of the time frame $\boldsymbol{TF}_i$. Here, we initialize $\boldsymbol{TF}_i$ with empty time slots, to allow "complexification" of the time frames and emergence of frames with a minimum number of transmit/listen actions.[3]

Algorithm 1 Distributed Hill Climbing algorithm used to optimize the time frame $\boldsymbol{TF}_i$ of node n_i in a network.

1: **procedure** DistributedHillClimbing
2: $\quad \boldsymbol{TF}_i \leftarrow$ initialize() ▷ Empty slots
3: $\quad f_i = \text{eval}(\boldsymbol{TF}_i)$ ▷ Local fitness
4: $\quad$ **while** True **do**
5: $\quad\quad \boldsymbol{TF}' \leftarrow \text{mutate}(\boldsymbol{TF}_i, mr)$ ▷ Mutation
6: $\quad\quad f' \leftarrow \text{eval}(\boldsymbol{TF}')$ ▷ Local fitness
7: $\quad\quad$ **if** $f' > f_i$ **then** ▷ Maximizing reward
8: $\quad\quad\quad f_i \leftarrow f'$
9: $\quad\quad\quad \boldsymbol{TF}_i \leftarrow \boldsymbol{TF}'$
10: $\quad\quad$ **end if**
11: $\quad$ **end while**
12: **end procedure**

2.2.1. Local fitness computation

During the evaluation process, each node executes its time frame as discussed in Section 2.1. The fitness value of each node is then computed locally, as a cumulative sum of scores. We refer to these scores as *reinforcements*. Each reinforcement is associated to the node behavior (i.e., a combination of action, queue and outcome) displayed in each time step of the evaluation. These reinforcements specify the rewards for the possible node behaviors. For instance, a transmit action would be more preferable if there is a packet in the queue of a node. However, if the queue of a node is empty, performing a transmit action would be unnecessary. Therefore, to encourage/discourage specific behaviors, it is possible to define rewarding (positive) or punishing (negative) reinforcements depending on the preference of the action in a certain situation.

The complete list of node behavior reinforcement associations (NBRAs) defined in our experiments is given in Table 1. For each node behavior, the corresponding reinforcement value r_k is provided as the reward $r(t)$ at time step t. These reinforcements are aggregated to compute the local fitness value f_i of each node n_i, as follows:

$$f_i = \begin{cases} C & \text{if } \boldsymbol{TF}_{i,j} \text{ is empty } \forall j \in \{1, \dots, S\}; \\ \sum_{t=1}^{K} r(t) & \text{otherwise.} \end{cases} \tag{1}$$

i.e., if all the time slots in $\boldsymbol{TF}_i$ are empty, we set f_i to a large negative constant value C, which is intended to encourage transmit/listen actions. Else, we compute f_i by aggregating the reinforcement values $r(t)$ obtained in each step t.

The NBRAs can be divided into two groups. The first group (ID: 2, 3, 5, 6, and 7) aims to encourage the reception and transmission of the packets in the network. The second group (ID: 1, 4, 8, and 9) aims to penalize unnecessary actions, thus implicitly minimizing the number of slots used for transmit/listen actions, a proxy for energy consumption.

Since each node tries to maximize the sum of its rewards, these reinforcements obviously play a crucial role in the optimization process. In principle, for the first group of behaviors the nodes should be rewarded/punished more severely since the actions corresponding to those behaviors directly affect the transmission of packets. On the other hand, the reinforcement signals for the second group can be relaxed, since as said they aim mainly at reducing unnecessary actions. However, it is especially important to find a balance on the reinforcement signals of the second group because if these rewards/punishments are too high the nodes will avoid actions which are supposedly unnecessary yet may lead to exploration of new connections and establishing of new pathways. On the other hand, if these rewards/punishments are too low, the nodes will perform too many unnecessary actions. Although this may lead to several established pathways, it would also increase the network energy consumption and yield to an unnecessarily high number of duplicate packets. For instance, punishing a node in the cases where it listens but nothing is received (behavior 8 and 9) would reduce the number of listen actions in the time frame. However, this would also reduce the probability of establishing possible new connections. Similarly, behavior 4 rewards a node if it remains idle when the queue is empty, yet other actions may be tried. To demonstrate that, we performed a sensitivity analysis to show the difference in results when different reinforcement values are used in behavior 4, 8, and 9, see Table 2 in the next Section.

2.2.2. Mutation operator

The DHC algorithm uses a mutation operator to perturb the time frame $\boldsymbol{TF}_i$ of each node. The mutation operator (see Line 5 in Algorithm 1) simply samples, for each time slot, a new value with a probability of mr (mutation rate). Sampling is performed by selecting with uniform probability a random action different from the one present in the time slot.

3. Experimental setup

In this section, we present our experimental setup: the proposed algorithm settings (Section 3.1), the network settings (Section 3.2), and the benchmark algorithms (Section 3.3). Our experiments have been performed on a Linux workstation with 32 GB RAM and CPU Intel i7 with multi-threading at the level of runs. The code is implemented in MATLAB.

3.1. Proposed algorithm settings

The exact reinforcement values used in the NBRAs (see r_k in Table 1) can be assigned differently. This in turn can change the rewards/punishments of certain actions. We tested the proposed algorithm for seven different NBRA assignments, given in Table 2. These particular assignments (in the following, referred to as "rules") were defined based on domain knowledge. These rules differ only for the

[2] Code available at: https://github.com/anilyaman/Evolution-of-Protocols

[3] On the other hand, random initialization is crucial in centralized approaches to obtain a diverse set of global network solutions that facilitate exploration and crossover.

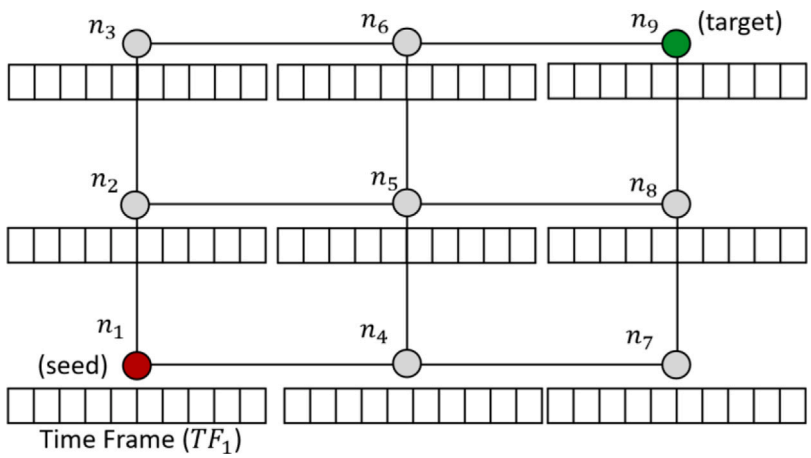


Fig. 2. Example network with nine nodes spatially distributed in a grid topology. Seed and target nodes are shown in red and green, respectively.

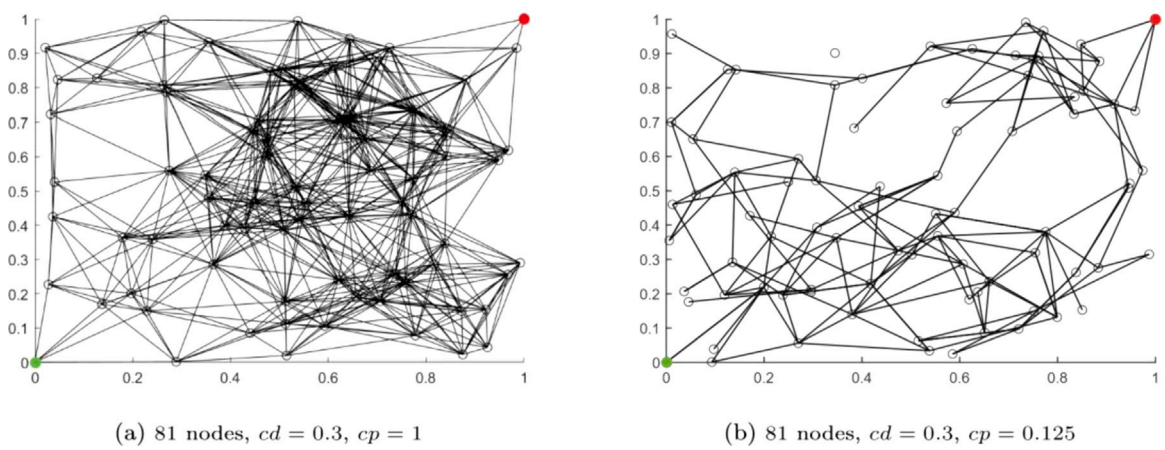


Fig. 3. Examples of random networks with 81 nodes. Connection distance (cd) and connection probability (cp) are used to control the connectivity of the networks. Seed (in location $(0,0)$) and target (in location $(1,1)$) nodes are shown in green and red, respectively.

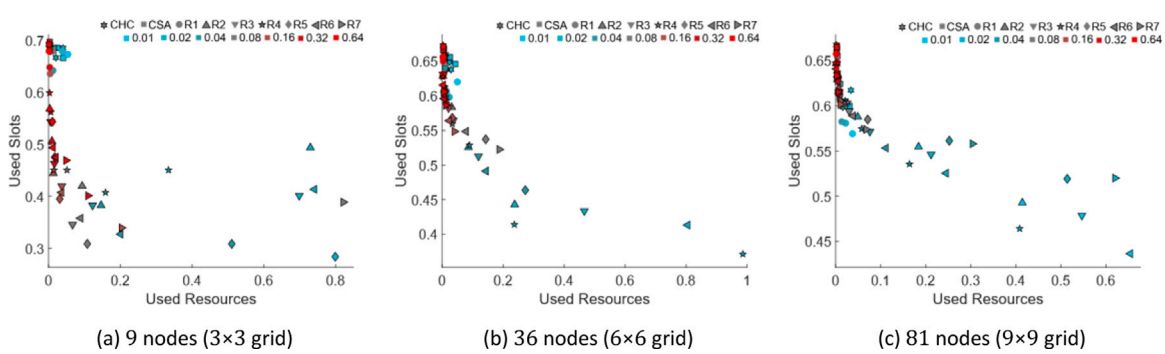


Fig. 4. Ratio of used slots vs ratio of used resources (median across 28 runs) of CHC, CSA and the proposed DHC algorithm (with seven different rules × seven different mutation rates) on grid topologies.

reinforcement values r_4, r_8 and r_9, to assess their effect on the minimization of the used slots. The other reinforcement values concern mainly the packet transmission, as discussed earlier in Section 2.2.

Among the tested rules, rule 1 is the most relaxed one in terms of unnecessary slot use since it does not provide any rewards/punishments for r_4, r_8 and r_9. Rules 2 and 3 reward only for r_4, to keep idle slots empty if they are not used. Rules 4 and 5 aim to provide punishments for unnecessary listen actions. Rules 6 and 7 provide both rewards and punishments for not using unnecessary idle slots and performing unnecessary listen actions, respectively.

We tested the proposed algorithm for seven mutation rate values following a geometric series, namely $mr = \{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64\}$. The maximum number of function evaluations was set to 10 000. However, to reduce the running time we stop the evolutionary process as soon as a solution that obtains 100% delivery rate of the packets to the target node is found (even though further refinement might be

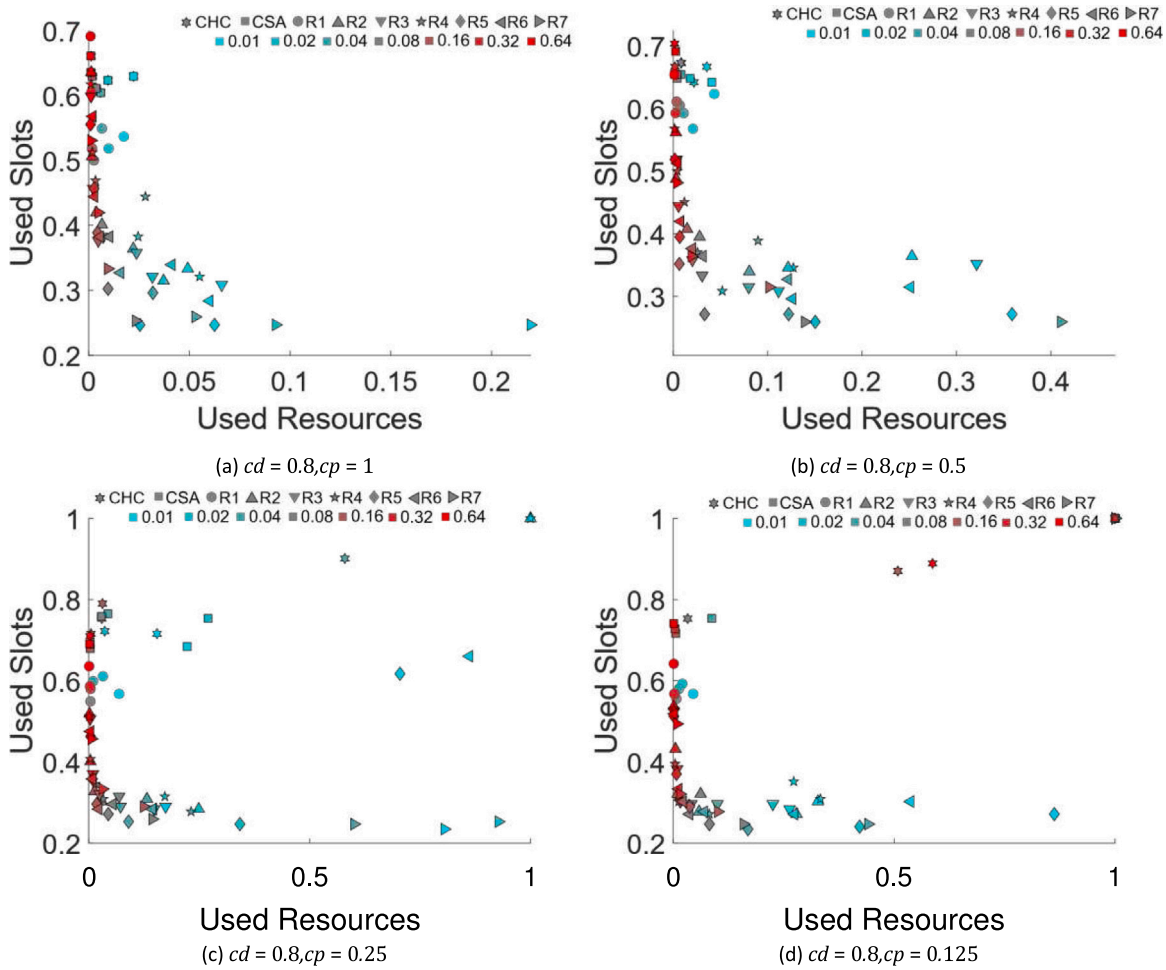


Fig. 5. Ratio of used slots vs ratio of used resources (median across 28 runs) of CHC, CSA and the proposed DHC algorithm (with seven different rules × seven different mutation rates) on random network configurations with 9 nodes.

Table 2
Rules for the node behavior reinforcement associations.

Rule ID	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
1	−1	1	−1	0	−1	1	1	0	0
2	−1	1	−1	0.5	−1	1	1	0	0
3	−1	1	−1	1	−1	1	1	0	0
4	−1	1	−1	0	−1	1	1	−0.5	−0.5
5	−1	1	−1	0	−1	1	1	−1	−1
6	−1	1	−1	0.5	−1	1	1	−0.5	−0.5
7	−1	1	−1	1	−1	1	1	−1	−1

possible). For each rule and mutation rate, the algorithm was executed for 28 independent runs.

3.2. Network settings

We tested the proposed algorithm on square (Manhattan-like) grids and randomly generated network topologies consisting of 9, 36, and 81 nodes.

The grid-like networks have been chosen to represent cases with a regular topology and a fixed number of connections per node (apart from the nodes at the border of the network, each node has $2(\sqrt{N}-1)$ connections). The random networks, instead, have been chosen to represent partially connected mesh networks, where the number of connections Both kinds of networks are representatives of widely adopted network topologies in real-world applications (Hekmat, 2006).

In the case of grid networks, the seed and target nodes are placed in opposite corners. An example grid network structure with 9 nodes is illustrated in Fig. 2.

In the case of random topologies, we considered a 2D Cartesian plane $[0,1]^2$ where the seed and target nodes are assigned to the coordinates $(0,0)$ and $(1,1)$, respectively, while the remaining $N-2$ nodes are assigned to random (x,y) coordinates. The connectivity of the random networks are adjusted based on two parameters, *connection distance* (cd) and *connection probability* (cp). A connection is established between two nodes when the Euclidean distance between them is smaller than cd, with probability cp: i.e., for all $i,j \in \{1,\dots,N\}$ where $i \neq j$, node n_i is connected to node n_j if $\text{dist}(n_i,n_j) < cd$ and $rand < cp$, where dist() is the Euclidean distance and $rand$ is a real valued uniform random variable in $[0,1]$. For 9, 36 and 81 nodes we set cd to 0.8, 0.5, and 0.3, respectively. For each network size, we considered four cp values, namely $cp = \{1, 0.5, 0.25, 0.125\}$. Therefore, in total we considered 12 random network configurations with various sizes and connectivity, see Fig. 3 for two examples. For each random network configuration we generated 28 networks and ran the algorithm once for each one independently.

For both kinds of networks (grid and random), we set the number of time slots S to be equal to the number of nodes N. During the evaluation process of the networks, $M = 5$ packets are generated in the seed node to be delivered to the target node. We set the number of steps K to $M \times S$.

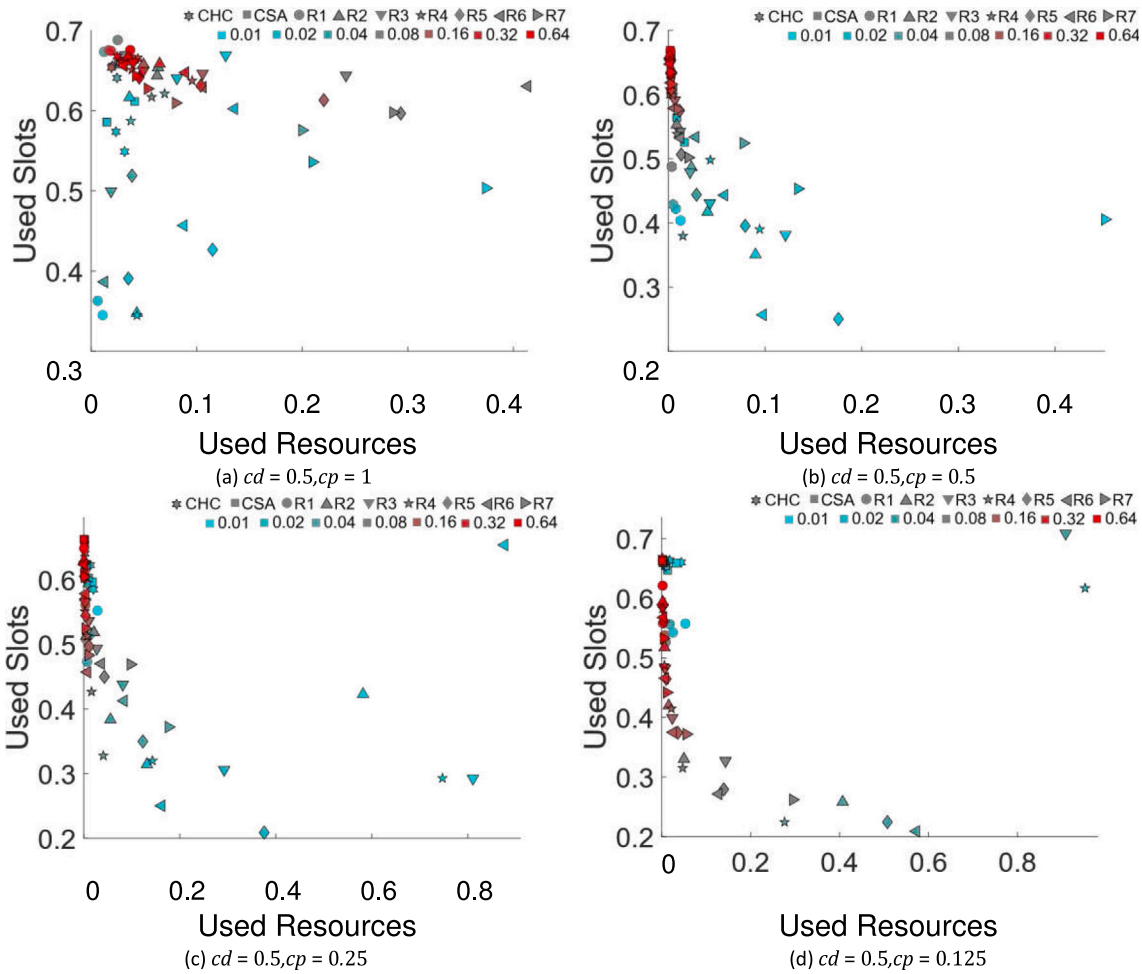


Fig. 6. Ratio of used slots vs ratio of used resources (median across 28 runs) of CHC, CSA and the proposed DHC algorithm (with seven different rules × seven different mutation rates) on random network configurations with 36 nodes.

3.3. Benchmark algorithms

To compare the results of our proposed approach, we considered seven centralized offline algorithms from the literature. For these algorithms, we conducted the optimization in a centralized (network-level) fashion, i.e., optimizing global network solutions obtained by concatenating the time frames of all nodes. The evaluation was performed as described in Section 2.1, in this case after deconstructing the global network solution into individual node time frames.

Contrarily to the proposed distributed approach, where each node locally maximizes its aggregated reward, see Eq. (1), the centralized algorithms do not make use of any online reinforcement during the fitness evaluation. Instead, they use *a posteriori* objective functions based on the global performance of the networks, namely the hop distance of the packets to the target node, and the node activity (used slots). The former is computed as follows:

$$f^{(1)} = \max_{i=1,\dots,M}\left(\min_{c=1,\dots,L_i}\left(h_i^c\right)\right) \tag{2}$$

where $h_i^c = \text{hop}(p_i^c, n_{target})$. Here, p_i^c indicates the cth copy of the ith packet and hop() measures the shortest hop distance from the node reached by p_i^c to the target node n_{target}, whereas L_i and M indicate, respectively, the number of copies of a given packet and the number of unique packets originating from the seed node (in our case $M = 5$). Finally, $f^{(1)}$ is scaled in $[0, 1]$ by dividing the maximum possible hop distance in a network (from any node to the target node).

For the latter, we simply find the average number of non-empty slots per node as follows:

$$f^{(2)} = \frac{1}{NS}\sum_{i=1}^{N}\sum_{j=1}^{S} A(i,j) \tag{3}$$

where:

$$A(i,j) = \begin{cases} 1, & \text{if } TF_{i,j} \text{ is “T” (transmit) or “L” (listen)}, \\ 0, & \text{otherwise.} \end{cases}$$

We divide the benchmark algorithms into three groups:

1. **Group 1 (Pareto multi-objective optimization)**: NSGA-II (Deb et al., 2002) and MSEA (Tian et al., 2019). The two algorithms were configured to perform Pareto optimization minimizing $f^{(1)}$ and $f^{(2)}$. For these experiments we used the implementation of NSGA-II and MSEA available in the PlatEMO platform (Tian et al., 2017), with the default parameter settings. The representation and evolutionary operators were adjusted to handle our ternary representation.
2. **Group 2 (scalarized multi-objective optimization)**: Centralized Hill Climbing with 2 Objectives (CHC2O), Centralized Simulated Annealing with 2 Objectives (CSA2O) and Genetic Algorithm with 2 Objectives (GA2O). In this group, we minimize the sum of two objectives as $f^{(1)} + f^{(2)} \in [0, 2]$. The implementation of CHC2O and CSA2O is the same as CHC and CSA (in Group 3), except for the use of the second objective. In the case of GA2O, we configured the algorithm with roulette wheel selection with 10 elites, 1-point crossover operator with 0.9 probability, and the mutation operator used in DHC.

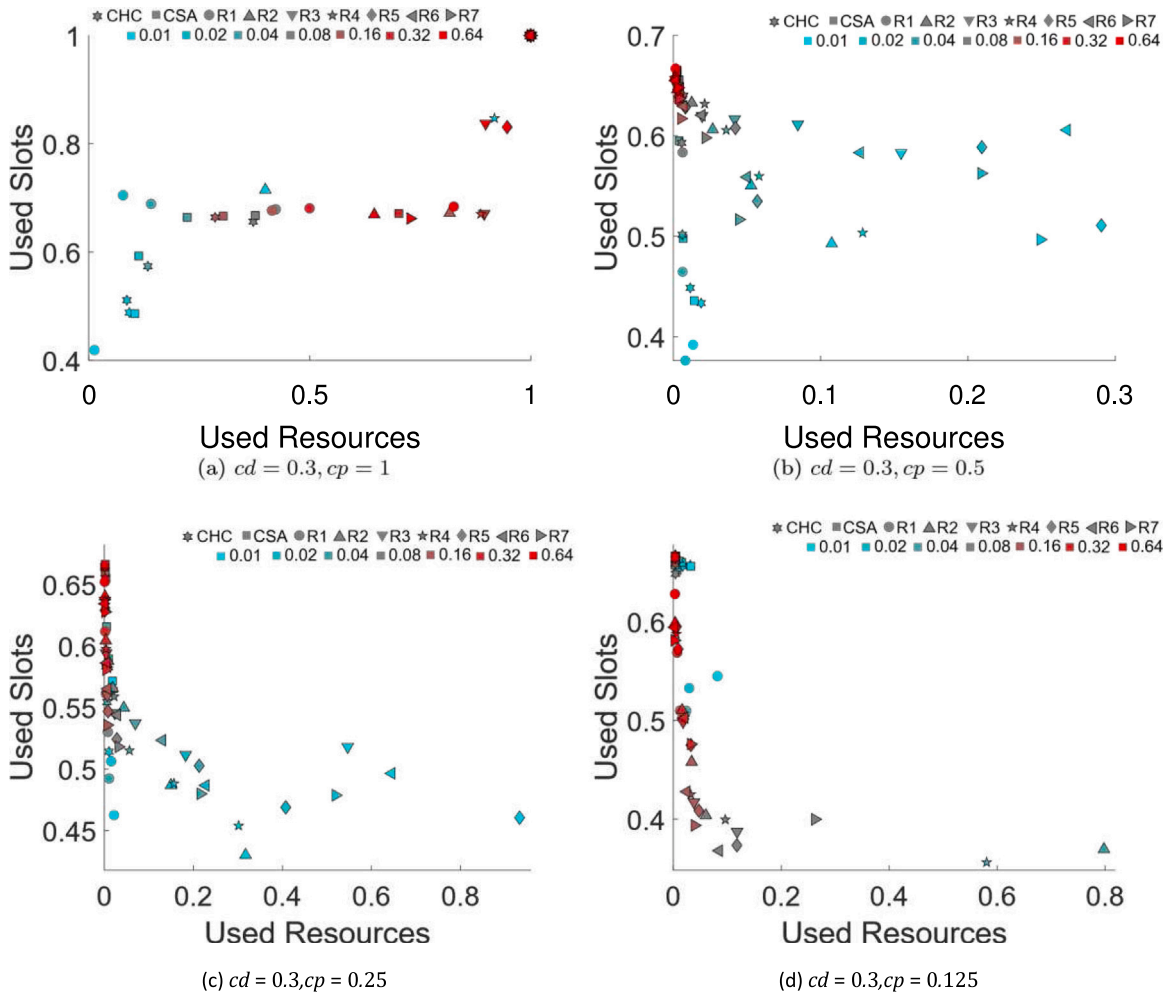


Fig. 7. Ratio of used slots vs ratio of used resources (median across 28 runs) of CHC, CSA and the proposed DHC algorithm (with seven different rules × seven different mutation rates) on random network configurations with 81 nodes.

3. **Group 3 (single-objective optimization)**: Centralized Hill Climbing (CHC) and Centralized Simulated Annealing (CSA) algorithms. The CHC and CSA iteratively perform mutations on a single centralized global network solution. In CHC, only solutions that are better or equal to the current solution are accepted for the next iteration, whereas CSA uses a temperature parameter T to accept worse solutions based on the probability $e^{(f_{new}-f_{old})/T}$. T is scheduled to be reduced at each iteration to decrease this probability ($T \leftarrow \alpha T, \alpha < 1$) (Kirkpatrick et al., 1983). In our experiments, we assign 0.9 for α. Both CHC and CSA minimize $f^{(1)}$.

The rationale for choosing the aforementioned algorithms is the following. The algorithms in the first group aim at finding an explicit trade-off between the two objectives of interest, i.e., the maximum distance to the target and the number of used slots, formulated as in Eqs. (2) and (3), respectively. Thus, the comparison with these algorithms should reveal if our method is able to find at least similar trade-offs, even though it does not explicitly search for trade-off solutions. For comparison, we chose to use NSGA-II, MSEA as they are well-known multi-objective algorithms for which public implementations are available. The algorithms in the second group aim instead at finding trade-offs based on a scalarized function (please remember that each of the two objectives in Eqs. (2) and (3) is scaled in $[0, 1]$, thus we do not use weights). Comparing our method with the algorithms in the second group should then give us insight on the capability of DHC to find good trade-offs between network performance and energy consumption w.r.t. approaches that, instead of performing Pareto optimization, solve a single-objective optimization problem. CHC, CSA and GA have been selected for being closely related to our proposed method (although they are, obviously, centralized). Moreover, GA was added to the comparison to show how a canonical population-based, single objective evolutionary algorithm can perform on this task. Finally, the algorithms in the third group are included in the comparison to show how CHC and CSA (chosen, again, for being closely related to our proposed method) perform when they are configured to optimize only $f^{(1)}$. This final comparison should rule out the effect of the scalarization applied to the algorithms in the second group.

For the population-based algorithms (NSGA-II, MSEA, and GA2O) we used a population of 50 solutions and we set the maximum number of function evaluations to 10 000 (without any early stop). For CHC and CSA (both with single and two objectives) we set 10 000, with early stop in case of 100% delivery rate solution found.

4. Experimental results

We analyze the numerical results obtained with the proposed algorithm from two different perspectives: scalability, i.e., we evaluate how the performance of DHC change depending on the network size (Section 4.1), and robustness, i.e., we evaluate how our method is able to cope with various kinds of network perturbations and show adaptive capabilities (Section 4.2).

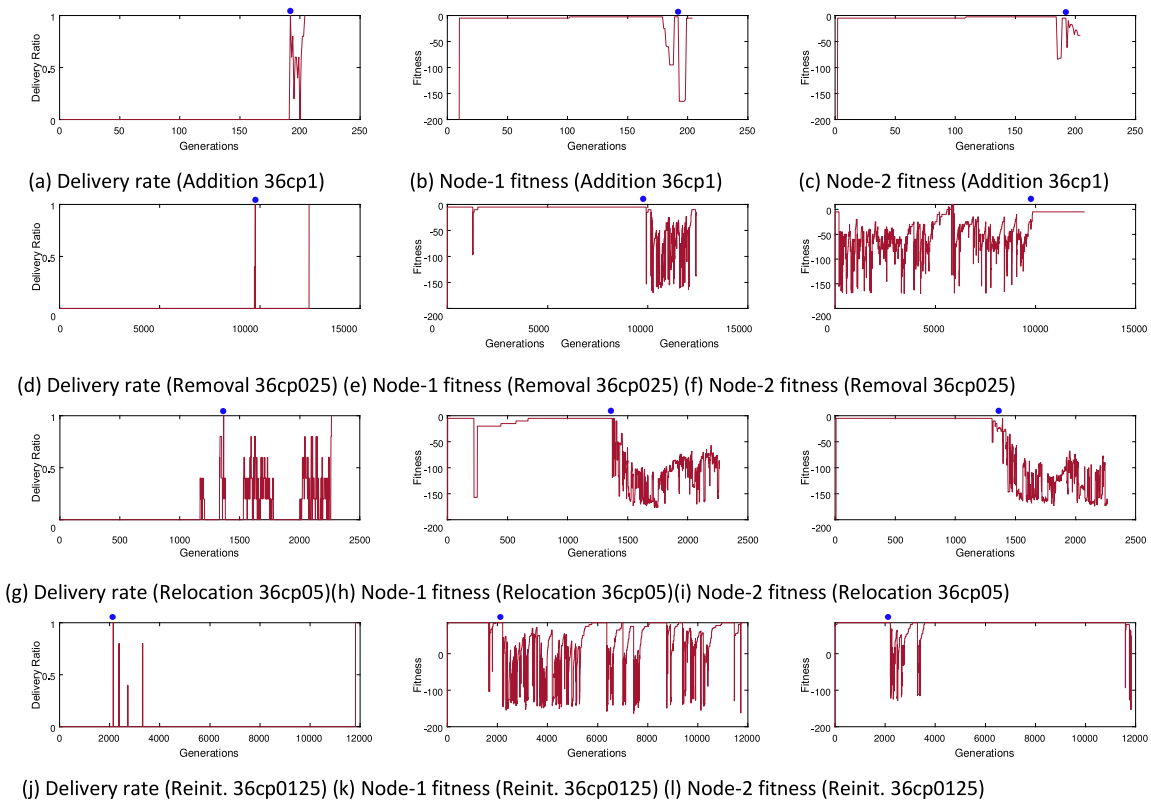


Fig. 8. Delivery rate of the network (first column) and fitness trend of two randomly selected nodes (second and third columns) during example evolutionary processes performed for the robustness experiments. The rows provide example processes for 36cp1, 36cp05, 36cp025 and 36cp0125 problem instances with addition, removal, relocation, and reinitialization perturbations respectively. The blue dots indicate the end of the evolutionary process before applying the perturbation (starting from the initial network until 100% delivery rate is achieved).

4.1. Evaluating scalability

The complete comparison of the results is given in Table 3. The results of DHC and Group 3 are the best results obtained across all mutation rates tested (see Section 4.3 for the parameter analysis). We should note that, except for the cases marked by "-", all the compared algorithms reach $f^{(1)} = 0$ (distance to target node), which yields 100% delivery rate. For this reason, we compare the algorithms on $f^{(2)}$ (slot use). In the case of Group 1, we select from the final Pareto front the solution with $f^{(1)} = 0$ and minimum $f^{(2)}$ for comparison. Once again, it is worth stressing that our method does not optimize explicitly nor $f^{(1)}$ or $f^{(2)}$, but it optimizes node-local rewards, see Eq. (1).

The results show the superiority in terms of scalability of DHC over the benchmark algorithms. The standard deviations and statistical analysis of the results are provided in e Appendix. The significance of the results is measured by the Wilcoxon rank-sum test (Wilcoxon, 1992) and the Nemenyi test (Demšar, 2006). When the size of the problem is low (i.e., in the case of 9 nodes), the algorithms based on a multi-objective optimization approach (in particular NSGA-II) and the ones that combines two objectives (i.e., GA2O, CHC2O and CSA2O) find better solutions in terms of ratio of used slots. On the other hand, while the size of the problem increases, DHC is able to find solutions with a smaller ratio of used slots relative to the other algorithms. We observe a further improvement in the performance of DHC when sparsity increases (i.e., cp decreases). However, the standard deviations of the results provided by DHC are relatively higher than those obtained by the multi-objective optimization algorithms, which might be due to the fact that, differently from these algorithms, DHC does not minimize explicitly the ratio of used slots.

4.1.1. Trade-off between slots and resources

Another important aspect to analyze is the effectiveness, in terms of function evaluations, of the proposed approach. Fig. 4 shows the results of CHC, CSA and the proposed DHC with the seven rules shown in Table 2 in terms of used resources and used slots in the case of grid topologies. Likewise, Figs. 5, 6, and 7 shows the results for the random network configurations with 9, 36, and 81 nodes. Here, *used resources* refer to the ratio of function evaluations needed to find a solution with 100% delivery rate, while *used slots* refer to the ratio of non-empty slots in the solution found.

Overall, CHC and CSA are able to find a solution quickly. However, they often find solutions with a large ratio of used slots. On the other hand, DHC is able to find solutions with a smaller ratio of used slots, although different reward/punishment assignments produce solutions with different ratios of used slots. In particular, we observe that rules which are more restrictive in terms of reward/punishment of unnecessary slot use (i.e., 5, 6, and 7) tend to find solutions with a smaller ratio of used slots. However, they tend to use more fitness evaluations. Furthermore, we observe that higher mutation rates provide more randomness, resulting in similar results w.r.t. the CHC and CSA algorithms. In those cases, the solutions are found quite quickly but the ratio of slot use is high. Slightly higher mutation rates appear to be helpful for more restrictive rules rather than for the less restrictive ones. Overall, the best performing mutation rate ranges in $0.01 - 0.04$. However, the exact mutation rate that performs the best varies across different network configurations and rules.

4.2. Evaluating robustness

We tested DHC in four scenarios which require adaptation of previously found solutions to new conditions of the network. These scenarios include addition, removal, relocation and reinitialization of the nodes in the network. In case of addition, 10 nodes were included into the networks at random locations, with random time frames. In case of removal, 6 randomly selected nodes were removed from the networks.

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.04035	0.0206	0.0134	0.0057	0.00465	0.0034	0.0014
CSA	0.03995	0.0255	0.0145	0.00735	0.0039	0.00345	0.0021
R1	0.0537	0.03845	0.0124	0.0116	0.0045	0.0033	0.0015
R2	1	0.7295	0.146	0.0933	0.01335	0.0094	0.00335
R3	1	0.6986	0.1222	0.0669	0.0369	0.0163	0.00595
R4	1	0.3341	0.1582	0.0513	0.0148	0.0063	0.00275
R5	1	0.7991	0.5104	0.1076	0.03115	0.0171	0.0108
R6	1	0.7409	0.2006	0.089	0.0361	0.0204	0.01235
R7	1	1	1	0.8215	0.2029	0.1096	0.0488

Mutation Rate (%)

(a) % Used resources (3×3 nodes)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.6852	0.6667	0.679	0.679	0.6914	0.679	0.6914
CSA	0.6667	0.6852	0.6852	0.679	0.6852	0.6975	0.6914
R1	0.6728	0.679	0.642	0.679	0.6358	0.6481	0.679
R2	1	0.4938	0.3827	0.4198	0.4444	0.5062	0.5679
R3	1	0.4012	0.3827	0.3457	0.4198	0.463	0.5432
R4	1	0.4506	0.4074	0.4506	0.4506	0.5617	0.5988
R5	1	0.284	0.3086	0.3086	0.3951	0.4753	0.5432
R6	1	0.4136	0.3272	0.358	0.4074	0.4753	0.4938
R7	1	1	1	0.3889	0.3395	0.4012	0.4691

Mutation Rate (%)

(b) % Used slots (3×3 nodes)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.02915	0.02435	0.01515	0.0118	0.0074	0.00605	0.0034
CSA	0.043	0.0262	0.01095	0.0077	0.00625	0.0034	0.0047
R1	0.0501	0.02365	0.01415	0.0083	0.0059	0.00365	0.0025
R2	1	0.2374	0.0864	0.03145	0.0111	0.00755	0.00265
R3	1	0.466	0.1193	0.0364	0.021	0.00905	0.0028
R4	0.9862	0.2366	0.08995	0.0335	0.01415	0.00775	0.00225
R5	1	1	0.2724	0.1431	0.03335	0.01355	0.00645
R6	1	0.8051	0.1443	0.0787	0.02325	0.00645	0.0036
R7	1	1	1	0.1872	0.04045	0.01215	0.0059

Mutation Rate (%)

(c) % Used resources (6×6 nodes)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.6381	0.6489	0.6385	0.6601	0.6516	0.6597	0.6732
CSA	0.6466	0.6566	0.6404	0.6566	0.6686	0.6671	0.6721
R1	0.6204	0.5984	0.6069	0.6119	0.6327	0.6501	0.6563
R2	1	0.4429	0.5258	0.5837	0.5972	0.61	0.6289
R3	1	0.4336	0.5127	0.5644	0.5826	0.5976	0.6312
R4	0.3711	0.414	0.5289	0.5594	0.586	0.6134	0.6277
R5	1	1	0.4637	0.5374	0.5683	0.5899	0.6042
R6	1	0.4132	0.4915	0.5486	0.564	0.5965	0.6157
R7	1	1	1	0.5228	0.5486	0.5864	0.6061

Mutation Rate (%)

(d) % Used slots (6×6 nodes)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.0339	0.02105	0.0159	0.009	0.0046	0.00215	0.0015
CSA	0.0313	0.02075	0.00925	0.00445	0.00295	0.00205	0.00175
R1	0.03735	0.02225	0.0135	0.0094	0.0046	0.0035	0.0012
R2	0.4143	0.1837	0.04895	0.0196	0.00695	0.0038	0.0014
R3	0.5465	0.2116	0.0759	0.03005	0.00765	0.0044	0.00155
R4	0.4083	0.1638	0.0578	0.0272	0.0081	0.00365	0.0015
R5	1	0.514	0.252	0.0711	0.0106	0.00535	0.0029
R6	0.6542	0.2449	0.1118	0.0386	0.00855	0.0054	0.00185
R7	1	0.6201	0.3034	0.06745	0.01055	0.00595	0.0023

Mutation Rate (%)

(e) % Used resources (9×9 nodes)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.6171	0.6055	0.5983	0.6087	0.6363	0.6629	0.6671
CSA	0.5998	0.6036	0.6238	0.6552	0.6616	0.666	0.6647
R1	0.5691	0.5807	0.5823	0.6123	0.6291	0.6481	0.6573
R2	0.4928	0.5547	0.5876	0.6016	0.6189	0.6343	0.6466
R3	0.4788	0.5463	0.5714	0.594	0.6155	0.6286	0.6462
R4	0.4642	0.5356	0.5745	0.6039	0.6116	0.6266	0.644
R5	1	0.5194	0.5614	0.5847	0.6047	0.614	0.6363
R6	0.4366	0.5256	0.5534	0.5887	0.6126	0.6267	0.6407
R7	1	0.5202	0.5581	0.5738	0.6014	0.6151	0.6331

Mutation Rate (%)

(f) % Used slots (9×9 nodes)

Fig. 9. Variation of the evolved TDMA MAC protocol performance for Centralized Hill Climbing (CHC), Centralized Simulated Annealing (CSA) and Distributed Hill Climbing (DHC) with 7 different rules on grid networks with 3×3, 6×6 and 9 × 9 nodes.

In case of relocation, 18 randomly selected nodes were reassigned to new randomly initialized locations. In the case of reinitialization, the time frame of 18 randomly selected nodes were reinitialized randomly.

The experimental process involves first finding solutions for the initial networks through the proposed distributed optimization process, then applying each of the aforementioned perturbations, independently. After each perturbation, we establish links based on the cp and cd parameters, as described in Section 3.2. Then, we run the optimization process for 10 000 evaluations to find solutions adapted to the new conditions. Examples of delivery rate and fitness trends of the nodes during the evolutionary processes before and after the perturbations can be found in Section 4.2.1.

Table 4 shows the results of DHC after perturbations. The "Used resources" sub-table indicates the median of the ratio of the function evaluations used to find a 100% delivery rate solution for the perturbed networks, while the "Used slots" sub-table indicates the ratio of occupied slots in the corresponding solutions. "Initial solution" indicates the median of the ratio of used resources and median of ratio of used slots found for the initial network prior to the perturbation. We tested the algorithm on random networks with 36 nodes, using the best rules found in Table 3 (shown in bold for each network configuration).

Concerning the resource use, we observe that the algorithm can find solutions to the perturbed networks in a short time, usually using less than 10% of the allocated function evaluations. However, adaptation

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.0222	0.00955	0.0058	0.00375	0.00185	0.00115	0.0012
CSA	0.0222	0.00955	0.0058	0.00375	0.00185	0.00115	0.0012
R1	0.0174	0.0099	0.00655	0.0026	0.00165	0.00125	0.0009
R2	0.04915	0.03715	0.0221	0.00655	0.00375	0.00165	0.00125
R3	0.066	0.0316	0.0237	0.0072	0.00475	0.0022	0.00115
R4	0.05505	0.0245	0.0281	0.00585	0.00335	0.0018	0.00105
R5	0.06245	0.0254	0.03185	0.0096	0.00435	0.00265	0.0009
R6	0.05995	0.04125	0.0157	0.01035	0.00565	0.00285	0.002
R7	0.2193	0.0928	0.0529	0.02335	0.0095	0.0048	0.00115

Mutation Rate (%)

(a) % Used resources (cd=0.8, cp=1)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.6296	0.6235	0.6049	0.6111	0.6296	0.6358	0.6605
CSA	0.6296	0.6235	0.6049	0.6111	0.6296	0.6358	0.6605
R1	0.537	0.5185	0.5494	0.5	0.5185	0.6049	0.6914
R2	0.3333	0.3148	0.3642	0.4012	0.4198	0.5062	0.6358
R3	0.3086	0.321	0.358	0.3827	0.3765	0.4568	0.5988
R4	0.321	0.3827	0.4444	0.3827	0.4691	0.5123	0.6173
R5	0.2469	0.2469	0.2963	0.3025	0.3889	0.4568	0.5556
R6	0.284	0.3395	0.3272	0.3827	0.3827	0.4444	0.5679
R7	0.2469	0.2469	0.2593	0.2531	0.3333	0.4198	0.5309

Mutation Rate (%)

(b) % Used slots (cd=0.8, cp=1)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.0355	0.0219	0.00855	0.008	0.0036	0.00185	0.00145
CSA	0.04085	0.01775	0.00805	0.00395	0.00405	0.0015	0.0026
R1	0.0434	0.02085	0.01095	0.00695	0.0036	0.00255	0.00125
R2	0.2528	0.1216	0.08055	0.02785	0.0146	0.00305	0.0028
R3	0.3214	0.1114	0.08	0.03055	0.0203	0.00565	0.0035
R4	0.1272	0.05185	0.08975	0.02565	0.01175	0.00425	0.00125
R5	0.3589	0.1504	0.1222	0.03315	0.00645	0.00685	0.0017
R6	0.2511	0.1263	0.1217	0.03125	0.0201	0.00775	0.00475
R7	1	0.9708	0.4105	0.1393	0.1015	0.0193	0.0039

Mutation Rate (%)

(c) % Used resources (cd=0.8, cp=0.5)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.6667	0.642	0.6728	0.6728	0.6543	0.6667	0.7037
CSA	0.642	0.6481	0.6543	0.6481	0.6543	0.6543	0.6914
R1	0.6235	0.5679	0.5926	0.6049	0.6111	0.5926	0.6543
R2	0.3642	0.3457	0.3395	0.3951	0.4074	0.4877	0.5617
R3	0.3519	0.3086	0.3148	0.3333	0.358	0.4444	0.5185
R4	0.3457	0.3086	0.3889	0.3704	0.4506	0.5	0.5679
R5	0.2716	0.2593	0.2716	0.2716	0.3519	0.3951	0.5185
R6	0.3148	0.2963	0.3272	0.3642	0.3765	0.4198	0.5123
R7	1	0.6543	0.2593	0.2593	0.3148	0.3642	0.4815

Mutation Rate (%)

(d) % Used slots (cd=0.8, cp=0.5)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.1545	0.03625	0.5797	0.02945	0.0311	0.00505	0.0029
CSA	0.2228	0.2704	0.04335	0.02815	0.0037	0.00275	0.0017
R1	0.0687	0.03225	0.00975	0.0039	0.00415	0.00275	0.00095
R2	1	0.249	0.1318	0.02935	0.01325	0.00375	0.0018
R3	0.174	0.1481	0.07195	0.0687	0.0247	0.00965	0.003
R4	1	0.1723	0.2316	0.0329	0.015	0.0037	0.00155
R5	0.7045	0.3421	0.0902	0.0443	0.018	0.00865	0.0041
R6	0.8614	0.1459	0.1444	0.0548	0.0226	0.0087	0.00375
R7	0.8051	0.9274	0.6006	0.1438	0.1246	0.03125	0.00615

Mutation Rate (%)

(e) % Used resources (cd=0.8, cp=0.25)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	0.716	0.7222	0.9012	0.7531	0.7901	0.716	0.7099
CSA	0.6852	0.7531	0.7654	0.7593	0.679	0.6914	0.6914
R1	0.5679	0.6111	0.5988	0.5494	0.5802	0.5864	0.6358
R2	1	0.284	0.3086	0.3025	0.3272	0.4012	0.5185
R3	0.2901	0.284	0.2901	0.3148	0.321	0.3704	0.5062
R4	1	0.3148	0.2778	0.3086	0.3457	0.4074	0.5123
R5	0.6173	0.2469	0.2531	0.2716	0.2963	0.358	0.463
R6	0.6605	0.284	0.284	0.2963	0.284	0.358	0.4753
R7	0.2346	0.2531	0.2469	0.2593	0.2901	0.3333	0.4568

Mutation Rate (%)

(f) % Used slots (cd=0.8, cp=0.25)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	1	1	1	0.0325	0.5082	1	0.587
CSA	1	1	0.08685	1	0.0056	0.0038	0.0018
R1	0.0451	0.02055	0.01285	0.00705	0.00355	0.00225	0.00115
R2	0.3275	0.2789	0.0585	0.06195	0.0096	0.0049	0.0011
R3	0.2632	0.2255	0.09985	0.0408	0.0158	0.00865	0.00125
R4	0.2726	0.3332	0.08145	0.03565	0.0123	0.0042	0.00145
R5	0.8624	0.4217	0.1692	0.08185	0.0374	0.00725	0.0014
R6	0.5369	0.2741	0.0701	0.03665	0.02225	0.01225	0.00235
R7	1	1	0.4408	0.159	0.1011	0.0152	0.00845

Mutation Rate (%)

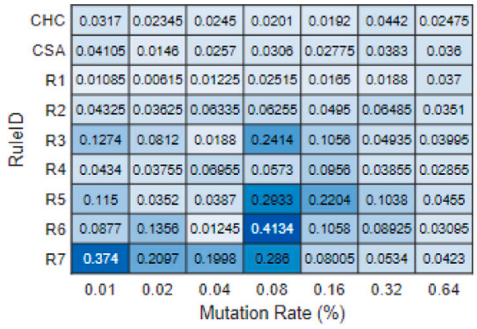
(g) % Used resources (cd=0.8, cp=0.125)

RuleID	0.01	0.02	0.04	0.08	0.16	0.32	0.64
CHC	1	1	1	0.7531	0.8704	1	0.8889
CSA	1	1	0.7531	1	0.716	0.7284	0.7407
R1	0.5679	0.5926	0.5802	0.5556	0.5679	0.5679	0.642
R2	0.3025	0.2716	0.2778	0.321	0.321	0.4321	0.537
R3	0.284	0.2963	0.2963	0.2963	0.3025	0.3827	0.5247
R4	0.3519	0.3086	0.2716	0.2963	0.3025	0.3951	0.5309
R5	0.2716	0.2407	0.2346	0.2469	0.2901	0.3704	0.5123
R6	0.3025	0.2716	0.2778	0.2716	0.3025	0.3333	0.5185
R7	1	1	0.2469	0.2469	0.2778	0.321	0.4938

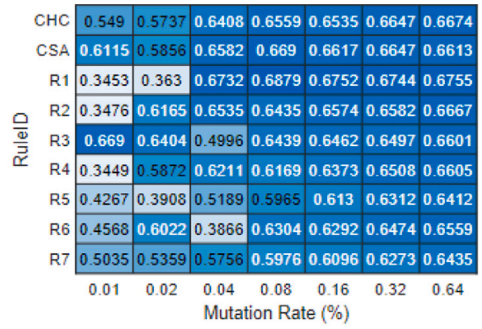
Mutation Rate (%)

(h) % Used slots (cd=0.8, cp=0.125)

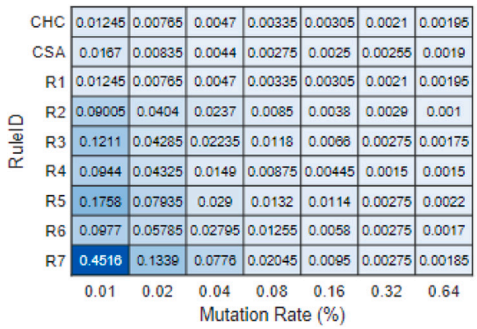
Fig. 10. Variation of the evolved TDMA MAC protocol performance for Centralized Hill Climbing (CHC), Centralized Simulated Annealing (CSA) and Distributed Hill Climbing (DHC) with 7 different rules on random networks with 9 nodes and various levels of connection distance (cd) and connection probability (cp).



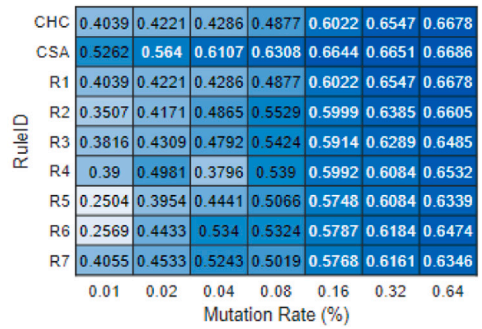
(a) % Used resources (cd=0.5, cp=1)



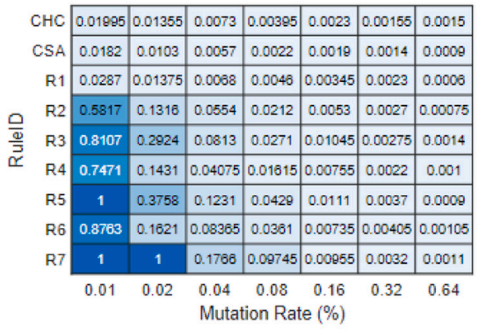
(b) % Used slots (cd=0.5, cp=1)



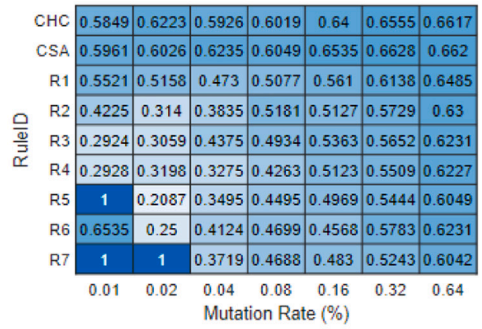
(c) % Used resources (cd=0.5, cp=0.5)



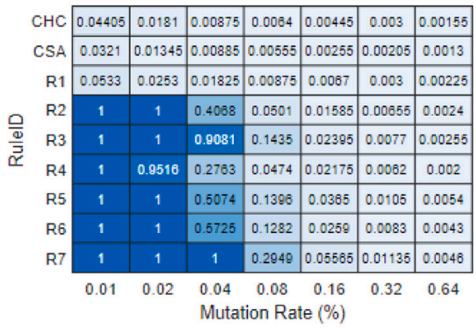
(d) % Used slots (cd=0.5, cp=0.5)



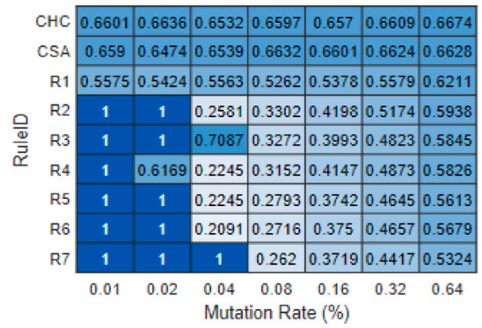
(e) % Used resources (cd=0.5, cp=0.25)



(f) % Used slots (cd=0.5, cp=0.25)



(g) % Used resources (cd=0.5, cp=0.125)



(h) % Used slots (cd=0.5, cp=0.125)

Fig. 11. Variation of the evolved TDMA MAC protocol performance for Centralized Hill Climbing (CHC), Centralized Simulated Annealing (CSA) and Distributed Hill Climbing (DHC) with 7 different rules on random networks with 36 nodes and various levels of connection distance (cd) and connection probability (cp).

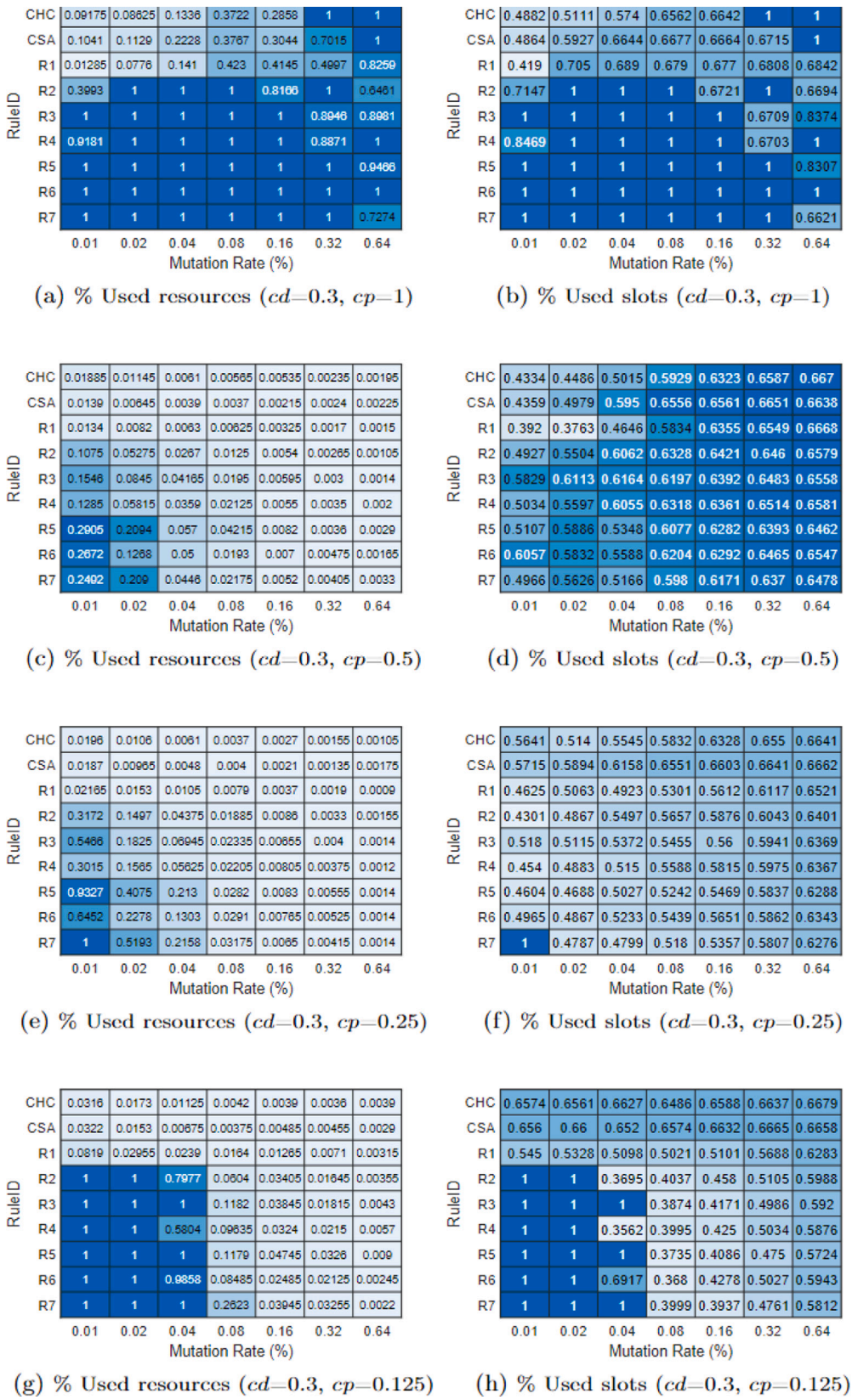


Fig. 12. Variation of the evolved TDMA MAC protocol performance for Centralized Hill Climbing (CHC), Centralized Simulated Annealing (CSA) and Distributed Hill Climbing (DHC) with 7 different rules on random networks with 81 nodes and various levels of connection distance (cd) and connection probability (cp).

Table 3
Comparison of the algorithms on grid and random networks with 9 to 81 nodes. The problem size is computed as N (no. of nodes) $\times$ S (no. of slots), which yields to N^2 since we set $N = S$. The value in each cell shows the ratio of used slots (median across 28 runs at the end of the optimization process) for the solutions found by the algorithms ("–" indicates that no solution is found).

Problem	Size	Group 1		Group 2			Group 3		DHC						
		NSGA-II	MSEA	GA2O	CHC2O	CSA2O	CHC	CSA	R1	R2	R3	R4	R5	R6	R7
Grid9	81	**0.10**	0.23	**0.10**	0.17	0.15	0.66	0.66	0.64	0.38	0.35	0.41	0.28	0.33	0.34
9cp1	81	**0.07**	0.22	**0.07**	**0.07**	**0.07**	0.60	0.60	0.50	0.31	0.31	0.32	0.25	0.28	0.25
9cp05	81	**0.07**	0.23	**0.07**	**0.07**	**0.07**	0.64	0.64	0.57	0.34	0.31	0.31	0.26	0.30	0.26
9cp025	81	–	–	**0.07**	0.10	0.10	0.71	0.69	0.55	0.28	0.28	0.28	0.25	0.28	0.23
9cp0125	81	–	–	**0.07**	**0.07**	0.12	0.75	0.73	0.56	0.27	0.28	0.27	0.23	0.27	0.25
Grid36	1296	0.43	0.55	0.49	**0.31**	**0.31**	0.63	0.64	0.60	0.44	0.43	0.37	0.46	0.41	0.52
36cp1	1296	0.45	0.56	0.54	0.38	0.38	0.55	0.59	0.35	0.35	0.50	**0.34**	0.39	0.39	0.50
36cp05	1296	0.43	0.56	0.49	0.33	0.33	0.40	0.53	0.40	0.35	0.38	0.38	**0.25**	0.26	0.41
36cp025	1296	0.42	0.58	0.48	0.32	0.31	0.58	0.60	0.47	0.31	0.29	0.29	**0.21**	0.25	0.37
36cp0125	1296	0.41	0.60	–	0.57	0.55	0.66	0.65	0.53	0.26	0.33	0.22	0.22	**0.21**	0.26
Grid81	6561	0.59	0.62	0.58	0.52	0.52	0.59	0.59	0.57	0.49	0.48	0.46	0.52	**0.44**	0.52
81cp1	6561	0.66	0.65	0.63	0.63	0.64	0.49	0.49	**0.42**	0.67	0.67	0.67	0.83	–	0.66
81cp05	6561	0.60	0.62	0.59	0.55	0.56	0.43	0.44	**0.38**	0.49	0.58	0.50	0.51	0.56	0.50
81cp025	6561	0.59	0.64	0.58	0.52	0.52	0.51	0.57	0.46	**0.43**	0.51	0.45	0.46	0.49	0.48
81cp0125	6561	0.59	0.64	–	0.59	0.58	0.66	0.65	0.50	0.37	0.39	**0.36**	0.37	0.37	0.39

Table 4
Results found by DHC (median across 28 runs) on the initial network ("Initial solution") and after perturbing the initial network by addition, removal, relocation, and reinitialization of nodes (each perturbation is considered independently).

Problem	Used resources				
	Initial solution	Adaptation after perturbation			
		Addition	Removal	Relocation	Reinit.
36cd05cp1	0.04	0.27	0.01	0.05	0.03
36cd05cp05	0.17	0.02	0.06	0.07	0.02
36cd05cp025	0.37	0.06	0.58	0.26	0.06
36cd05cp0125	0.57	0.25	0.35	0.59	0.86
Problem	**Used slots**				
	Initial solution	Adaptation after perturbation			
		Addition	Removal	Relocation	Reinit.
36cd05cp1	0.34	0.64	0.49	0.64	0.67
36cd05cp05	0.25	0.49	0.32	0.33	0.53
36cd05cp025	0.21	0.37	0.21	0.21	0.54
36cd05cp0125	0.21	0.36	0.20	0.20	0.40

requires more evaluations in the case of the network with the highest sparsity ($cp = 0.125$).

Concerning the ratio of used slots, this seems to increase in general after perturbations. However, we observe that removal and relocation do not appear to have a high impact, especially in sparse networks. On the other hand, addition and reinitialization produces an increased slot use in all networks. This is likely due to the random reinitialization of the time frames occurring during the addition and reinitialization processes.

4.2.1. Runtime behavior of the evolutionary process

Fig. 8 shows the delivery rate of the networks and the fitness trends of two randomly selected nodes during the evolutionary processes performed for the robustness experiments. For example, in the experiment with the addition perturbation (shown in the first row), 100% delivery rate is achieved around the 200th generation with the initial network. Then, the perturbation decreases the delivery rate. However, DHC is able to recover from the perturbation and achieve 100% delivery rate again in about 30 generations.

4.3. Parameter analysis

We report in the heatmaps shown in Figs. 9–12 the variation of the evolved TDMA MAC protocol performance (as % of used resources and used slots, average values across 28 runs of each algorithmic setting at the end of the optimization process, i.e., either as soon as a viable protocol configuration capable to obtain 100% delivery rate is found, or after 10000 evaluations) w.r.t. the mutation rate (%) and the rule ID used in DHC, as well as the two baseline single-objective centralized algorithms, namely Centralized Hill Climbing (CHC) and Centralized Simulated Annealing (CSA), for the cases of grid networks (9, 36 and 81 nodes), as well as random networks with 9, 36, and 81 nodes, respectively. For the random networks, we consider different combinations of connection distance (cd) and connection probability (cp) values. In the heatmaps, darker (lighter) color means worse (better) performance.

The main findings that can be inferred from the figures can be summarized as follows:

- Concerning the used resources (first column of each figure), apart from the case of random networks with 81 nodes and cd=0.3, cp=1 (Fig. 12(a)), in all the tested settings low mutation rates tend to lead to higher % resource consumption. This is particularly evident in the grid topologies when DHC is configured to use rules from R2 to R7. Another interesting case is random networks with 9 nodes and cd=0.8, cp=0.125 (Fig. 10(g)), where CHC and CSA perform quite poorly for the lowest values of mutation rate. Apart from the two aforementioned peculiar cases, when higher mutation rates are used the resource consumption tends to be lower, with most of the algorithms reaching roughly similar values.
- Concerning the used slots (second column of each figure), in most cases there is a trend similar to that observed for the used resources (the lower the mutation rate, the worse the performance), with some exceptions represented by e.g. the random networks with higher values of cp, especially with 36 and 81 nodes (Figs. 11(b), 11(d), 12(b), and 12(d)). Overall, apart from these four cases DHC appears to use in general less slots than both centralized approaches.

Apart from these two trends, this analysis reveals that in general the optimal mutation rate depends on the network size and the specific DHC rule adopted for the online protocol evolution. This observation suggests, for instance, a possible extension of DHC to use (self-)adaptive mutation rates.

5. Conclusions

We proposed a decentralized, online evolutionary optimization algorithm, referred to as Distributed Hill Climbing (DHC), for optimizing a TDMA MAC protocol on a distributed network of nodes. In the proposed approach, each node evolves its time frame locally. To do that, we devised a set of reinforcement rules to assign predefined scores

to the actions of the nodes and compute their local fitness accordingly. Overall, we tested seven different reinforcement rules. We found that the DHC algorithm was able to evolve, in all the tested network scenarios, efficient TDMA MAC protocols with %100 delivery rate. Moreover, even though Distributed Hill Climbing was not configured to explicit minimize the node activity (i.e., the number of used time slots, that can be seen as a proxy for energy consumption), different reinforcement rules allowed the emergence of various protocols with different quality in terms of node activity.

We compared our algorithm with seven centralized single and multi-objective approaches, in which the optimization is performed at a global network level concatenating the time frames of all nodes in the network. In the multi-objective cases, we introduced a second objective function to minimize explicitly the node activity. Based on our comparisons, the benchmark algorithms with the explicit second objective showed better performance in terms of node activity only on the low-dimensional scenarios. On the other hand, DHC showed better performance when the network size increased.

Furthermore, the evaluation of the scalability the proposed algorithms revealed another important finding: in fact, the DHC algorithm was able to obtain the best performances (in terms of energy consumption) especially in the case of larger, sparser networks, typically requiring less function evaluations to converge. Moreover, we observed that the proposed algorithm is able to quickly adapt the protocol to network perturbations such as node addition, removal, relocation and reinitialization. These two features (scalability and robustness), which derive from the decentralized nature of our method, represent its main advantages.

One possible disadvantage of our method is the necessity of predefining reinforcement rules (such as those presented in Table 2, which as we have seen affect the overall evolution of the TDMA protocol. Finding these reinforcement association rules may be difficult in some specific contexts.

Therefore, one possible direction for future investigations will be to extend the algorithm to other rules. Another possibility would be to apply it to different network layers, where online adaptation might provide an even greater benefit. Moreover, it would be interesting to compare the proposed method against other, more recent metaheuristics for single-objective optimization, e.g., based on Swarm Intelligence (Heidari et al., 2019; Li et al., 2020; Wang et al., 2019; Yang et al., 2021), metaphor-less methods, such as the one proposed in Ahmadianfar et al. (2021), or methods that make use of information feedback models (Wang & Tan, 2019). Finally, verifying the proposed protocols in hardware would add further experimental value to the proposed method.

CRediT authorship contribution statement

Anil Yaman: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Tim van der Lee:** Methodology, Validation, Writing – review & editing. **Giovanni Iacca:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data not applicable. The code is accessible publicly.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2022.118627.

References

Ahmadianfar, I., Heidari, A. A., Gandomi, A. H., Chu, X., & Chen, H. (2021). RUN beyond the metaphor: an efficient optimization algorithm based on Runge Kutta method. *Expert Systems with Applications*, *181*, Article 115079.

Akhshabi, S., & Dovrolis, C. (2011). The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *SIGCOMM conference* (pp. 206–217). ACM.

Al Dallal, J., & Saleh, K. A. (2012). Synthesizing distributed protocol specifications from a UML state machine modeled service specification. *Journal of Computer Science and Technology*, *27*(6), 1150–1168.

Aloi, G., Bedogni, L., Felice, M. D., Loscri, V., Molinaro, A., Natalizio, E., Pace, P., Ruggeri, G., Trotta, A., & Zema, N. R. (2014). STEM-net: an evolutionary network architecture for smart and sustainable cities. *Transactions on Emerging Telecommunications Technologies*, *25*(1), 21–40.

Alouf, S., Carreras, I., Miorandi, D., & Neglia, G. (2007). Embedding evolution in epidemic-style forwarding. In *International conference on mobile adhoc and sensor systems* (pp. 1–6). IEEE.

Alouf, S., Neglia, G., Carreras, I., Miorandi, D., & Fialho, Á. (2010). Fitting genetic algorithms to distributed on-line evolution of network protocols. *Computer Networks*, *54*(18), 3402–3420.

Alsheikh, M. A., Lin, S., Niyato, D., & Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *Communications Surveys & Tutorials*, *16*(4), 1996–2018.

Birman, K., Constable, B., Hayden, M., Hickey, J., Kreitz, C., Van Renesse, R., Rodeh, O., & Vogels, W. (2000). The horus and ensemble projects: Accomplishments and limitations. In *DARPA information survivability conference and exposition. Vol. 1* (pp. 149–161). IEEE.

Bouabene, G., Jelger, C., Tschudin, C., Schmid, S., Keller, A., & May, M. (2009). The autonomic network architecture (ANA). *Journal on Selected Areas in Communications*, *28*(1), 4–14.

Bredeche, N., Haasdijk, E., & Prieto, A. (2018). Embodied evolution in collective robotics: A review. *Frontiers in Robotics and AI*, *5*, 12.

Bredeche, N., Montanier, J.-M., Liu, W., & Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, *18*(1), 101–129.

Büchi, J. R., & Landweber, L. H. (1990). Solving sequential conditions by finite-state strategies. In *The collected works of J. Richard Büchi* (pp. 525–541). Springer.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(Jan), 1–30.

Dovrolis, C. (2008). What would darwin think about clean-slate architectures? *SIGCOMM Computer Communication Review*, *38*(1), 29–34.

Eiben, A. E., Haasdijk, E., & Bredeche, N. (2010). Embodied, on-line, on-board evolution for autonomous robotics. In *Symbiotic multi-robot organisms: reliability, adaptability, evolution* (pp. 361–382). Springer.

El-fakih, K., Yamaguchi, H., & Bochmann, G. (1999). A method and a genetic algorithm for deriving protocols for distributed applications with minimum communication cost. *Journal of Physics A - Mathematical General*, 863–868.

Ergen, S. C., & Varaiya, P. (2010). TDMA scheduling algorithms for wireless sensor networks. *Wireless Networks*, *16*(4), 985–997.

Finkbeiner, B., & Gölz, P. (2017). Synthesis in distributed environments. arXiv:1710.05368.

Förster, A., & Murphy, A. L. (2011). Machine learning across the WSN layers. In *Emerging communications for wireless sensor networks*. IntechOpen.

Guo, K., & Lv, Y. (2020). Optimizing routing path selection method particle swarm optimization. *International Journal of Pattern Recognition and Artificial Intelligence*, Article 2059042.

Haasdijk, E., Bredeche, N., & Eiben, A. E. (2014). Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. *PLoS One*, *9*(6), 1–14.

Hajiaghajani, F., & Biswas, S. (2015a). Feasibility of evolutionary design for multi-access MAC protocols. In *Global communications conference* (pp. 1–7). IEEE.

Hajiaghajani, F., & Biswas, S. (2015b). MAC protocol design using evolvable state-machines. In *International conference on computer communication and networks* (pp. 1–6). IEEE.

He, H., Jin, S., Wen, C.-K., Gao, F., Li, G. Y., & Xu, Z. (2019). Model-driven deep learning for physical layer communications. *Wireless Communications*, *26*(5), 77–83.

Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, *97*, 849–872.

Hekmat, R. (2006). *Ad-hoc networks: fundamental properties and network topologies*. Springer Science & Business Media.

Holzmann, G. J. (1991). *Design and validation of computer protocols. Vol. 512*. Prentice Hall.

Holzmann, G. J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, *23*(5), 279–295.

Hutchinson, N. C., & Peterson, L. L. (1991). The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, *17*(1), 64–76.

Iacca, G. (2013). Distributed optimization in wireless sensor networks: an island-model framework. *Soft Computing*, *17*(12), 2257–2277.

Imai, P., & Tschudin, C. (2010). Practical online network stack evolution. In *International conference on self-adaptive and self-organizing systems* (pp. 34–41). IEEE.

Johnson, D. M., Teredesai, A. M., & Saltarelli, R. T. (2005). Genetic programming in wireless sensor networks. In *European conference on genetic programming* (pp. 96–107). Springer.

Kalmbach, P., Zerwas, J., Babarczi, P., Blenk, A., Kellerer, W., & Schmid, S. (2018). Empowering self-driving networks. In *Afternoon workshop on self-driving networks* (pp. 8–14).

Keller, A., Hossmann, T., May, M., Bouabene, G., Jelger, C., & Tschudin, C. (2008). A system architecture for evolving protocol stacks. In *International conference on computer communications and networks* (pp. 1–7).

Kellerer, W., Basta, A., & Blenk, A. (2015). Flexibility of networks: a new measure for network design space analysis? arXiv:1512.03770.

Kellerer, W., Kalmbach, P., Blenk, A., Basta, A., Reisslein, M., & Schmid, S. (2019). Adaptable and data-driven softwarized networks: Review, opportunities, and challenges. *Proceedings of the IEEE*, *107*(4), 711–731.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kulkarni, R. V., Forster, A., & Venayagamoorthy, G. K. (2010). Computational intelligence in wireless sensor networks: A survey. *Communications Surveys & Tutorials*, *13*(1), 68–96.

Lee, J.-H., & Cho, S. H. (2017). Tree TDMA MAC algorithm using time and frequency slot allocations in tree-based WSNs. *Wireless Personal Communications*, *95*(3), 2575–2597.

Lee, T. V. D., Exarchakos, G., & Groot, S. H. D. (2020). Distributed reliable and energy-efficient scheduling for LR-WPANs. *ACM Transactions on Sensor Networks*, *16*(4), 1–20.

Lewis, T., Fanning, N., & Clemo, G. (2006). Enhancing IEEE802.11 DCF using genetic programming. In *Vehicular technology conference. Vol. 3* (pp. 1261–1265). IEEE.

Li, S., Chen, H., Wang, M., Heidari, A. A., & Mirjalili, S. (2020). Slime mould algorithm: A new method for stochastic optimization. *Future Generation Computer Systems*, *111*, 300–323.

Miorandi, D., & Yamamoto, L. (2008). Evolutionary and embryogenic approaches to autonomic systems. In *International conference on performance evaluation methodologies and tools* (pp. 1–12).

Nakano, T. (2010). Biologically inspired network systems: A review and future prospects. *IEEE Transactions on Systems, Man, and Cybernetics*, *41*(5), 630–643.

Peshkin, L., & Savova, V. (2002). Reinforcement learning for adaptive routing. In *International joint conference on neural networks. Vol. 2* (pp. 1825–1830). IEEE.

Prieto, A., Bellas, F., Trueba, P., & Duro, R. J. (2016). Real-time optimization of dynamic problems through distributed embodied evolution. *Integrated Computer-Aided Engineering*, *23*(3), 237–253.

Roohitavaf, M., Zhu, L., Kulkarni, S., & Biswas, S. (2018). Synthesizing customized network protocols using genetic programming. In *Genetic and evolutionary computation conference companion* (pp. 1616–1623).

Saleh, K. (1996). Synthesis of communications protocols: an annotated bibliography. *SIGCOMM Computer Communication Review*, *26*(5), 40–59.

Sharples, N., & Wakeman, I. (2000). Protocol construction using genetic search techniques. In *Workshops on real-world applications of evolutionary computation* (pp. 235–246). Springer.

Siyari, P., Dilkina, B., & Dovrolis, C. (2017). Emergence and evolution of hierarchical structure in complex systems. In *Dynamics on and of complex networks* (pp. 23–62). Springer.

Stampa, G., Arias, M., Sánchez-Charles, D., Muntés-Mulero, V., & Cabellos, A. (2017). A deep-reinforcement learning approach for software-defined networking routing optimization. arXiv:1709.07080.

Su, Y., & Van Der Schaar, M. (2010). Dynamic conjectures in random access networks using bio-inspired learning. *Journal on Selected Areas in Communications*, *28*(4), 587–601.

Sun, Y., Zhang, Z., Li, X., Xiao, S., & Tang, W. (2020). An extensible frame structure for time division multiple access medium access control in vehicular ad-hoc networks. *Transactions on Emerging Telecommunications Technologies*.

Tao, N., Baxter, J., & Weaver, L. (2001). A multi-agent, policy-gradient approach to network routing. In *International conference on machine learning*.

Tekken-Valapil, V., & Kulkarni, S. S. (2017). Derivation of network reprogramming protocol with Z3. arXiv:1709.06604.

Tian, Y., Cheng, R., Zhang, X., & Jin, Y. (2017). PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. *IEEE Computational Intelligence Magazine*, *12*(4), 73–87.

Tian, Y., He, C., Cheng, R., & Zhang, X. (2019). A multistage evolutionary algorithm for better diversity preservation in multiobjective optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, *51*(9), 5880–5894.

Tschudin, C., & Yamamoto, L. (2005). Self-evolving network software. *Praxis Der Informationsverarbeitung Und Kommunikation*, *28*(4), 206–210.

Valencia, P., Lindsay, P., & Jurdak, R. (2010). Distributed genetic evolution in WSN. In *International conference on information processing in sensor networks* (pp. 13–23). ACM/IEEE.

Van Belle, W., Mens, T., & D'Hondt, T. (2003). Using genetic programming to generate protocol adaptors for interprocess communication. In *International conference on evolvable systems* (pp. 422–433). Springer.

Vardi, M. Y. (2018). The siren song of temporal synthesis. In *International conference on concurrency theory*.

Wadhwa, S., Rani, S., Verma, S., Shafi, J., & Wozniak, M. (2022). Energy efficient consensus approach of blockchain for IoT networks with edge computing. *Sensors*, *22*(10), 3733.

Wang, G.-G., Deb, S., & Cui, Z. (2019). Monarch butterfly optimization. *Neural Computing and Applications*, *31*(7), 1995–2014.

Wang, G.-G., & Tan, Y. (2019). Improving metaheuristic algorithms with information feedback models. *IEEE Transactions on Cybernetics*, *49*(2), 542–555. http://dx.doi.org/10.1109/TCYB.2017.2780274.

Weise, T., Geihs, K., & Baer, P. A. (2007). Genetic programming for proactive aggregation protocols. In *International conference on adaptive and natural computing algorithms* (pp. 167–173). Springer.

Weise, T., & Tang, K. (2011). Evolving distributed algorithms with genetic programming. *IEEE Transactions on Evolutionary Computation*, *16*(2), 242–265.

Weise, T., Zapf, M., & Geihs, K. (2008). Evolving proactive aggregation protocols. In *European conference on genetic programming* (pp. 254–265). Springer.

Wetherall, D. J., Guttag, J. V., & Tennenhouse, D. L. (1998). ANTS: A toolkit for building and dynamically deploying network protocols. In *Open architectures and network programming* (pp. 117–129). IEEE.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics* (pp. 196–202). Springer.

Wolpert, D., Tumer, K., & Frank, J. (1999). Using collective intelligence to route internet traffic. In *Advances in neural information processing systems* (pp. 952–960).

Woźniak, M., Sikora, A., Zielonka, A., Kaur, K., Hossain, M. S., & Shorfuzzaman, M. (2021). Heuristic optimization of multipulse rectifier for reduced energy consumption. *IEEE Transactions on Industrial Informatics*, *18*(8), 5515–5526.

Woźniak, M., Siłka, J., Wieczorek, M., & Alrashoud, M. (2020). Recurrent neural network model for IoT and networking malware threat detection. *IEEE Transactions on Industrial Informatics*, *17*(8), 5583–5594.

Xiao, Y. (2016). *Bio-inspired computing and networking*. CRC Press.

Yamamoto, L., Schreckling, D., & Meyer, T. (2007). Self-replicating and self-modifying programs in fraglets. In *Workshop on bio-inspired models of network, information and computing systems* (pp. 159–167). IEEE.

Yamamoto, L., & Tschudin, C. (2005). Genetic evolution of protocol implementations and configurations. In *International workshop on self-managed systems and services* (pp. 34–2007070218786). IFIP/IEEE.

Yaman, A., & Iacca, G. (2021). Distributed embodied evolution over networks. *Applied Soft Computing*, *101*, Article 106993.

Yang, Y., Chen, H., Heidari, A. A., & Gandomi, A. H. (2021). Hunger games search: Visions, conception, implementation, deep analysis, perspectives, and towards performance shifts. *Expert Systems with Applications*, *177*, Article 114864.

Yun, C., Cho, A.-R., Kim, S.-G., Park, J.-W., & Lim, Y.-K. (2013). A hierarchical time division multiple access medium access control protocol for clustered underwater acoustic networks. *Journal of Information and Communication Convergence Engineering*, *11*(3), 153–166.

Zhang, X., Li, J., Qiu, R., Mean, T.-S., & Jin, F. (2020). Optimized routing model of sensor nodes in internet of things network. *Sensors and Materials*, *32*(8), 2801–2811.