ISBN 978-83-60810-51-4

Experimental Analysis of Different Pheromone Structures in an Ant Colony Optimization Algorithm in Robotic Skin Design

Cristiano Nattero, Massimo Paolucci, Davide Anghinolfi, Fulvio Mastrogiovanni and Giorgio Cannata.
Università degli Studi di Genova
LIDO Lab @ DIST
13, Via Opera Pia
16145 Genova, Italy
Email: <name>.<surname>@unige.it

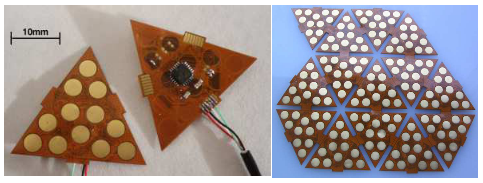
***Abstract*—The optimization of the wire routing in an artificial skin for robots consists in selecting a subset of links between adjacent tactile sensors in order to connect them to a finite set of microcontrollers with a minimum cost. The problem has been modelled as a minimum cost *Constrained Spanning Forest* problem with solution-dependent costs on arcs. The problem is NP-hard. A MIP formulation, which applicability is limited, is given. An *Ant Colony Optimization* (ACO) algorithm, recently introduced, is also described for tackling large scale problems. This paper introduces several different alternative pheromone structures, whose effectiveness is evaluated through experimental tests, performed on both real and synthetically generated instances.**

I. Introduction

To EQUIP robots, especially humanoids, with tactile sensing, the development of *robotic skin* is an active field of research. Artificial skins are devices composed of tactile sensors linked in a network of connections, and are researched because they would allow new kind of interaction, possibly safer and more natural, with people, other robots and the environment. The kind of technology used for the sensors allow the recognition of different types of information, especially pressure, proximity or temperature. Designing a robotic skin is a hard engineering task as it requires to deal with different conflicting requirements including resolution, speed, bandwidth, weight, consumption, placement, reliability, calibration, to name the main ones. In the last phases of the design, it is necessary to choose how to route the wires in the skin, in order to satisfy a set of requirements. A general optimization technique has been recently developed [1]: this article extends the contribution by exploring several algorithmic features.

The considered skin [2]–[4] is based on capacitive-technology and is formed by elements (Fig. 1a) which tile (Fig. 1b) the surface of a robot part, forming a so-called *patch*.

A data network is embedded in the elements: thanks to I/O ports on their sides, information can be sent to and forwarded from adjacent elements or, with use of additional wires, a microcontroller. Given a patch, an appropriate routing of the networking of the skin elements is required, in order to link each element to a microcontroller which will read its data. Each microcontroller can be connected to a limited number C of elements. The resulting wiring represents one of the major limiting factors to the implementation of large scale robot skin. Therefore, in order to reduce wiring complexity, each microcontroller can be connected directly to only one element of a patch, called the *entry point*, whereas the signals of the other sensing elements are routed via neighbouring skin elements. Note that each skin element can route the signals managed by a single microcontroller.



(a) (b)

Fig. 1: A skin element, bottom and up (a), and a patch (b) [2]–[4], (pictures courtesy of IIT, Italian Institute of Technology).

Beyond the minimization of routing complexity cost discussed above, two other main objectives must be considered in the skin layout: power consumption (mostly related to the number of microcontrollers) and the fault tolerance of each skin patch. The former is achieved by reducing the number of microcontrollers to the strictly necessary minimum to control all of the elements. *Fault tolerance* prescribes that in case of failure the system must avoid to suddenly stop working. On the contrary, a graceful performance degradation must be guaranteed, since residual functionalities can be essential to detect the failure and let the robot safely homing. The failure of a microcontroller is the most critical case and hence it is the main focus of this work. In such a case all the elements connected to the microcontroller become *blind*. Therefore, in order to reduce the damage it is advisable to uniformly distribute the load among microcontrollers. In addition, a significant performance degradation occurs if the blind elements form a big *sensing hole* on the patch; a strategy to mitigate this drawback consists in spreading the skin elements assigned to a microcontroller as much as possible.

The concepts explained above lead to the formulation of wire routing as an optimization problem with three distinct but interconnected aspects: (i) assign each skin element to a microcontroller; (ii) identify an entry point for each microcontroller; (iii) define wire routing.

The remainder of this paper is organized as follows. A review of the approaches to robotic skin optimization is given in Section II. Issues relevant to modelling this problem in the class of the *Constrained Spanning Forest* problems are discussed in Section III. A Mixed Integer Programming (MIP) formulation for the problem is introduced in Section IV. The proposed multi-start heuristic and the ant colony approaches are detailed in Section V. In Section VI the performance of the heuristic approaches are analysed and conclusions are drawn in Section VII.

II. Approaches to the Problem of Wiring in Robotic Skin Design

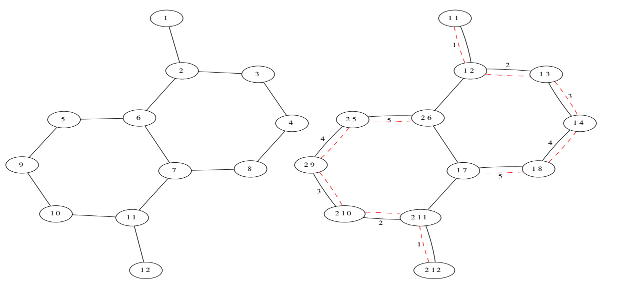
For a detailed review of tentative solutions to robotic skin optimization, the interested reader can refer to [1]. In the past two decades, research on tactile arrays for humanoid robots has been based on the design of grid-like wiring patterns. Solutions based on grid-like patterns are not efficient when scaling up to patches with hundreds of tactile sensors, since the number of wires quadratically increases with the array size. This problem is of the uttermost importance for highly embedded machines as humanoid robots, because the physical space necessary to host wires can be difficult to obtain. The approach based on telemetric robotic skin, aimed at avoiding wires at all by locating sensor chips inside a moulding material, which is drained within a suitable robot cover does not allow any control related to fault-tolerance requirements. Attempts to address the wire routing problem via skin modularity deeply compromise fault-tolerance. Deformable and stretchable skin sensors exploit the principle of Electrical Impedance Tomography to reconstruct pressure distribution over a given surface using only border measurements. Although this mechanism allows to avoid wiring the surface itself, the problem of the number of wires actually needed at the border (especially for large surfaces) is not addressed.

In spite of all the different approaches to the wiring problem available, it is important to note that a smart method to wiring taking into account fault-tolerance is currently missing. The only work which provides wire routing solutions taking into account both complexity and fault-tolerance is the authors recent paper [1], which presents two heuristic algorithms, one of which is an *Ant Colony Optimization*.

The main contribution of this article is an experimental investigation of five different pheromone structures for solving the wire routing problem.

III. A Graph-Based Model

Since the interconnections between robotic skin elements are fixed, it is convenient to represent a robotic skin patch as a graph $G = (V, E)$ where $V = \{v_0, \dots, v_n\}$ is a set of vertices and $E = \{e_0, e_1, \dots, e_m\}$ is a set of edges. V contains a vertex v for each skin element, whereas E includes an edge $e = (i, j)$ for each pair of neighbouring elements $i, j \in V, i < j$.



(a) (b)

Fig. 2: An example of a patch graph (a), and a possible solution (b).

An example of a graph associated with a patch is given in Fig. 2a. Note that the graph is supposed to be symmetric, so no direction is defined between two connected elements. Due to the characteristics of the considered robotic skin and independently from the elements shape, G is a planar graph. In addition, if the vertices associated with the skin elements on the border are neglected, then G is also a regular graph. Given a patch and a corresponding graph G, D_{ij} represents the Euclidean distance between the centroids of each pair of skin elements $(i, j) \in V \times V$. Finally, a set K of microcontrollers is given. In general, the minimum cardinality of K can be computed as $|K| = \left\lceil \frac{|V|}{C} \right\rceil$, where V is the set of skin elements and $\lceil x \rceil$ denotes the *ceiling* operator, which returns the smallest integer bigger than or equal to x.

The problem of optimally defining a wire routing for a patch of skin elements can be modelled within the class of *Constrained Spanning Forest* problems: the set of skin elements assigned to a microcontroller k corresponds to a set of vertices T_k, such that $|T_k| \leq C$, and the wiring defines an acyclic connected sub-graph $S_k(T_k, E_k)$ in G induced by T_k, that is a tree. Any vertex v in T_k can be chosen as root and used as entry point for the microcontroller k. The solution of the wiring problem is then a forest F corresponding to the the collection of all the trees S_k. Therefore, the connection of the skin elements in a patch to a set of microcontrollers and the consequent wire routing corresponds to a constrained spanning forest on the graph associated with the patch. As an example, Fig.2b shows a possible wiring for the patch graph of Fig.2a using two microcontrollers. In Fig.2b the node labels $(k|v)$ denote the index of the microcontroller followed by the index of the vertex, whereas the number associated with the edges represents the order of assignment of a connection to a microcontroller, allowing the identification of the *entry-points* (vertices 1 and 12, in the example) and the routing direction.

The *Maximum Weight Forest* (MaxWF) problem consists in defining a spanning forest that maximizes the sum of the weights associated with each edge. Although it is possible to solve MaxWF in polynomial-time with a greedy algorithm [5], the introduction of constraints makes the problem much more difficult. The *Maximum Weight t-restricted Forest* (MAXWCF(t)) problem, consisting in finding the MAXWF constrained to have no more than t edges in each tree, has been shown to be NP-hard [6]. The authors also give a $1/4$ approximating greedy algorithm. [7] deal with the *Minimum Weighted Constrained Spanning Forest* (MINWCF(p)) problem, which consists in computing the spanning forest of minimum weight such that every tree component contains at least p vertices. They demonstrate that the the problem is NP-hard, $\forall p \geq 4$ and [8] show that MINWCF(p) is NP-hard even with $p = 3$. [9] address the unweighted version of the same problem (MINCF(p)) and show that it is NP-hard for any $p \geq 4$, even on planar bipartite graphs of maximum degree 3. Moreover, they demonstrate that dropping the condition of planarity the problem becomes APX-hard for any $p \geq 3$, even on graphs with maximum degree 3. Despite some similarities, the problem faced in this paper differs both from MAXWF, MINWCF (and MINCF). As a matter of fact, the problem here considered is to find a constrained spanning forest F that serves all vertices, minimizing a cost function with two components: (i) the sum, on all the trees $S_k \in F$, of the absolute value of the difference between the number of vertices $|T_k|$ and the desired average number of vertices per tree, defined as $\lambda \triangleq \left\lceil \frac{|V|}{|K|} \right\rceil$; this cost component would favour the microcontroller load balancing; (ii) the sum, on all the trees $S_k \in F$ and on all the ordered pairs of vertices (i, j), such that $i < j$, $i, j \in T_k$, of the difference between the maximum distance between all the pairs of vertices in V and the distance between i and j; this component would favour the spreading of the skin elements associated with a single microcontroller, thus avoiding the risk that large skin spots could become blind in case of a microcontroller failure. Considering the second objective component, it is evident that, for the faced *Constrained Spanning Forest* problem, no cost can be *a priori* associated to an edge as it depends on which subsets of vertices in V are finally assigned the $|K|$ trees. Therefore, since the contribution of an edge to the objective function becomes known only after the introduction of that edge in a solution, a greedy heuristic appears not suitable for this problem. Nonetheless, if the contribution of each edge was known *a priori*, the considered problem could be reduced to a MINWCF. This allows to conclude that the problem is NP-hard.

IV. THE MATHEMATICAL FORMULATION

The wiring routing problem can be formulated as the problem of determining a spanning forest on the graph $G = (V, E)$ associated with a skin patch so that the forest is composed by at most $|K|$ trees and each tree includes at most C vertices. In addition, as for each microcontroller an entry point and a feasible routing must be determined, a root vertex and an orientation is selected for each tree so that the solution corresponds to a directed spanning forest consisting of a set of arborescences. This problem can be formulated as a mixed integer programming (MIP) problem considering the directed graph $G' = (V, A)$, where $A = \{(i, j), (j, i) : \forall (i, j) \in E\}$. In the following let $a = (i, j)$ denote a generic directed arc, where vertices i and j are respectively the tail and the head of a. Then, a feasible solution identifying a directed spanning forest H is a set of arborescences R_k, i.e., $H \triangleq \{R_k(T_k, A_k), k \in K\}$, where $T_k \subseteq V$ and $A_k \subseteq A$.

The following constants are used: D_{ij}, $\forall (i, j) \in V \times V$, Euclidean distance between vertices i and j; $D^{\max} = \max\{D_{ij}\}$, $D^{\min} = \min\{D_{ij}\}$; C, maximum number of vertices in a tree (microcontroller capacity); $\lambda \in (0, C]$, desired number of vertices in a tree (desired microcontroller load level); $w_1, w_2, w_3 \in \mathbb{R}_+$, weights modelling the priority of the objective function components.

The following decision variables are used:

- $r_{kv} \in \{0, 1\}$, $\forall k \in K, v \in V$; $r_{kv} = 1$ only if controller $k \in K$ is connected directly to vertex $v \in V$, that is, v is the root of tree k (the *entry point* for microcontroller k);
- $x_{ka} \in \{0, 1\}$, $\forall k \in K, a \in A$; $x_{ka} = 1$ only if $a \in A_k$; note that a vertex j is assigned to an arborescence (T_k, A_k) (i.e., it is controlled by microcontroller k) if either j is the root of k or it is the head of an arc $a = (i, j)$ such that $a \in A_k$;
- $y_{ij} \in \{0, 1\}$, $\forall (i, j) \in V \times V$, $i < j$; $y_{ij} = 1$ only if $i, j \in T_k$ for a given k; these variables determine if two vertices are assigned to the same arborescence and are used to evaluate the relative distance between vertices;
- $n_v \in \{0, 1\}$, $\forall v \in V$; $n_v = 1$ only if $v \notin T_k, \forall k \in K$; variables n_v are introduced to relax the vertex spanning condition, i.e., to model also the case of skin patches that generate graphs structured so that there is no possibility to assign all the vertices to a microcontroller (this point is discussed later in this Section);
- $\Delta_k \in \mathbb{R}_+$, $\forall k \in K$; these are deviational variables giving the absolute difference between $|T_k|$ and the average desired load λ for microcontrollers;
- $u_v \in \mathbb{R}$, $\forall v \in V$; variables used in *subtour elimination* constraints.

The problem is a multi-objective one requiring the minimization of the three following objectives:

$$O_1 \triangleq \sum_{v \in V} n_v \tag{1}$$

$$O_2 \triangleq \sum_{k \in K} \Delta_k \tag{2}$$

$$O_3 \triangleq \frac{1}{(D^{\max} - D^{\min})} \sum_{\substack{i,j \in V \\ i<j}} y_{ij} \left(D^{\max} - D_{ij} \right) \tag{3}$$

where O_1 (1) is the sum of vertices not assigned to any tree in the current solution, O_2 (2) is the sum over all microcontrollers of the variations from the average desired load λ and O_3 (3) corresponds to the sum of the normalized distances among every pair of vertices assigned to the same arborescence. The following three scaling factors, ν_1, ν_2 and ν_3, are introduced to ensure that $O_h \in [0, 1]$, $h = 1, 2, 3$:

$$\nu_1 \triangleq \frac{1}{|V|}; \quad \nu_2 \triangleq \frac{1}{|K|\lambda}; \quad \nu_3 \triangleq \frac{2}{[|V| (|V| - 1)]}; \tag{4}$$

Note that, as such factors are derived from worst case considerations, the scaled objectives could not reach the extremes of their variation range.

The multi-objective problem can be converted into a scalar minimization problem by combining the three objectives into a single weighted additive objective function (5), where the values of the weights are provided according to the different preference of the decision maker. A lexicographic priority can be imposed between any pair of objectives O_i, O_j, fixing $w_i \gg w_j$. Then, the proposed formulation is the following:

$$\min \quad w_1\nu_1 O_1 + w_2\nu_2 O_2 + w_3\nu_3 O_3 \tag{5}$$

subject to:

$$\sum_{k\in K}\Big(r_{kj} + \sum_{\substack{a\in A\\ a=(i,j)}} x_{ka}\Big) + n_j = 1 \qquad \forall j \in V \tag{6}$$

$$x_{ka} \le \sum_{\substack{f\in A\\ f=(j,i)}} x_{kf} + r_{ki} \quad \forall i \in V, k \in K, a=(i,l)\in A \tag{7}$$

$$\sum_{v\in V} r_{kv} \le 1 \qquad \forall k \in K \tag{8}$$

$$\sum_{j\in V}\sum_{\substack{a\in A\\ a=(i,j)}} \big(x_{ka} + r_{kj}\big) \le C \qquad \forall k \in K \tag{9}$$

$$u_i - u_j + C\sum_{k\in K} x_{ka} \le C-1 \qquad \forall a=(i,j)\in A \tag{10}$$

$$\sum_{j\in V}\sum_{a\in A}\big(x_{ka}+r_{kj}\big) - \lambda \le \Delta_k \qquad \forall k\in K \tag{11}$$

$$\sum_{j\in V}\sum_{a\in A}\big(x_{ka}+r_{kj}\big) - \lambda \ge -\Delta_k \qquad \forall k\in K \tag{12}$$

$$y_{ij} \ge \sum_{\substack{a_1\in A\\ a_1=(h,i)}} x_{ka_1} + r_{ki} + \sum_{\substack{a_2\in A\\ a_2=(l,j)}} x_{ka_2} + r_{kj} - 1 \tag{13}$$
$$\forall i,j \in V, k\in K, i<j$$

Constraints (6) require that each vertex must be at most assigned to a single arborescence (i.e., a microcontroller), being either the entry point for the arborescence or the head of one of its arcs. Constraints (7) ensure that an arc with origin in vertex i can be assigned to controller k only if another arc assigned to k enters in i or if i is an entry point for k. Constraints (8) impose that for each microcontroller there must be at most one entry point. Constraints (9) limit the number of vertices assigned to each microcontroller. Constraints (10) are the sub-tour elimination constraints introduced by [10]; here these constraints ensure that each microcontroller is associated with an acyclic connected and directed sub-graph, i.e., an arborescence. Constraints (11) and (12) define the unbalance between the number of vertices assigned to a microcontroller k and the average desired load. Constraints (13) impose $y_{ij} = 1$ if vertices i and j are controlled by the same microcontroller.

Variables n_v are necessary to determine feasible solutions for the graphs whose vertices cannot be partitioned into $|K|$ distinct arborescences, i.e., assigned to the available $|K|$ microcontrollers. Such situations may arise in case of patches generating a non connected graph, but also in case of connected graph with particular structures [1].

V. The Ant Colony Optimization Algorithm

Since the MIP model *as-is* is almost useless to solve large scale instances [1], a metaheuristic approach had to be developed and, being the problem instances very structured, the choice has been oriented towards an algorithm which makes use of learning. In particular, an *Ant Colony Optimization* (ACO) algorithm [11], together with an efficient *Candidate Strategy* (CS), has been implemented [1]. The ACO algorithm described in this Section shares the same structure of the algorithm described by Anghinolfi and Paolucci [12], which borrows concepts both by the *Ant Colony System* (ACS) [13] and the *Max-Min Ant System* (MMAS) [14]. The algorithm is outlined in Pseudocode 1.

Pseudocode 1 ACO algorithm.

```
Input: $G = (V,E)$, $|K|$, $\lambda, P$
 1: $F^* \leftarrow \emptyset$, $z_{\mathtt{best}} \leftarrow +\infty$
 2: $\pi \leftarrow$ initPheromone($G$)
 3: while termination_condition not met do
 4:     for all ant $a$ do
 5:         $F_a \leftarrow$ build solution( $G$, $|K|$, $\lambda$, $\pi$ )
 6:         if $z(F_a) < z_{\text{best}}$ then
 7:             $F^* \leftarrow F_a$
 8:             $z_{\mathtt{best}} \leftarrow z(F^*)$
 9:         end if
10:         localPheromoneUpdate($F_a$, $\pi$)
11:     end for
12:     globalPheromoneUpdate($F^*$, $\pi$)
13: end while
14: return $F^*$
```

The graph G, the number $|K|$ of microcontrollers, the desired occupancy level λ and the number P of artificial ants are fed into the algorithm. The best solution F^* and the best value $z_{\mathtt{best}}$ are initialized respectively to an empty set and to $+\infty$ (l. 1). The pheromone π is also initialized (l. 2). The main loop of the algorithm (l. 3 - 13) consists in making each ant (l. 4 - 11) construct a forest F_a which possibly spans all the vertices in V (l. 5), comparing the solution against the incumbent (l. 6) and eventually save it (l. 7 - 8). Pheromone trails are locally updated after each construction (l. 10) and globally updated at the end of the iteration (l. 12). The best solution found is finally returned (l. 14).

The solution construction proceeds incrementally [1], each ant a constructing a forest F_a one tree T at a time. A tree is constructed by adding an edge e to T. The edge e is obtained from an element z, which is extracted out of a set θ of candidates using information associated with pheromone trails. The set θ is generated from scratch before the insertion of each edge and Subsections V-A and V-B explain how to obtain it. The set θ can contain one among two possible types of elements: edges or vertices. A vertex v is called *free* if it has not been assigned to any tree yet so, If z corresponds to an edge $e = (i,j)$, either $i \in T$ and j is free, or $j \in T$ and i is free. In the other case, if z corresponds to a vertex v, then it is free. For the sake of completeness, any edge e connecting v to a vertex in T can be added, since their contribution to the objective function (5) is equivalent. The construction of current tree ends when the desired occupancy level has been reached ($|T| = \lambda$) or it is no longer possible to find an element for which the insertion is feasible ($\theta = \{\emptyset\}$).

Note that, for comparison purposes, it is possible to design a randomized constructive heuristic simply by neglecting the use of pheromone. In this case, the selection of the next element $z \in \theta$ to insert in T is performed randomly with a uniform probability distribution. A *Multi Start Heuristic* (MSH) is then obtained by restarting the construction. Conceptually, this corresponds to setting $P = 1$ but, in practice, it is convenient to write a dedicated implementation. For a fair comparison, the termination condition both of MSH and ACOs is the maximum running time.

Differently from MSH, when more alternatives exist, the ant performs the selection in two steps. First, as in the standard ACS, the ant chooses randomly the node selection rule between *exploitation* and *exploration* with probability $q \in [0, 1]$ and $1 - q$ respectively. Let $\pi(z)$ denote the pheromone trail associated with element z: the *exploitation* rule selects z deterministically according to

$$w = \arg\max_{w \in \theta} \pi(w) \tag{14}$$

and the *exploration* rule selects z according to a selection probability $p(z)$ obtained as

$$p(z) = \frac{\pi(z)}{\sum_{w \in \theta} \pi(w)} \tag{15}$$

Following the same approach proposed in [12], the pheromone trails do not depend on the cost function values associated with previously explored solutions. Rather, they vary in an arbitrary range $[\pi_{\min}, \pi_{\max}]$ such that $\pi_{\min} < \pi_{\max}$ and $\pi_{\min} \geq 0$. In this way, $\pi_{\min}$ and $\pi_{\max}$ are no longer parameters to be tuned. This feature makes the algorithm more independent from the specific problem or instance. After any ant a completes the construction of a solution F, in order to reduce the probability that the successive ants construct an identical solution during the same iteration, the pheromone trails are locally updated (l. 10) as follows

$$\pi(z) \leftarrow (1 - \rho)\,\pi(z), \quad \forall z \in F \tag{16}$$

where $\rho \in [0, 1]$ is the local evaporation parameter. The *global pheromone update* (l. 12 in Pseudocode 1) is performed in three steps:

(i) the perturbations due to the local pheromone updates are removed;

(ii) pheromone trails are evaporated as follows

$$\pi(z) \leftarrow (1 - \alpha)\,\pi(z), \quad \forall z \notin F^* \tag{17}$$

where F^* is the incumbent solution and $\alpha \in [0, 1]$ is the global evaporation parameter;

(iii) pheromone trails relevant to F^* are reinforced as follows

$$\pi(z) \leftarrow \pi(z) + \alpha \cdot \Delta\pi(z), \quad \forall z \in F^* \tag{18}$$

where $\Delta\pi(z)$ is the maximum pheromone reinforcement obtained as

$$\Delta\pi(z) = \pi_{\max} - \pi(z), \quad \forall z \in F^* \tag{19}$$

As shown in [12], these rules make the pheromone reach both bounds asymptotically.

A. Candidate Strategy

The set θ of elements candidate for insertion, is constructed, coherently with the used pheromone structure, upon a set ψ of candidate vertices, defined as follows. Let $A(v)$ be the set of vertices adjacent to v and $N(v) \subseteq A(v)$ be the set of free vertices adjacent to v. With a minor abuse in the notation, let also $A(T)$ be the sets of vertices adjacent to those in T and $N(T)$ the set of free vertices adjacent to those in T. Given a incomplete forest F, possibly empty, a first set ψ_0 is obtained including including all vertices in $N(T)$. If the tree is empty, then any free vertex is added to ψ_0 to bootstrap its construction. Now, ψ_0 can be passed directly to an ant for construction but this choice leads to poor quality solutions which, most of the times, are also incomplete. A first improvement is obtained by constructing a set $\psi_1 \subseteq \psi$ keeping only those vertices for which $|N(v)|$ is minimal, i.e.,

$$\psi_1 = \left\{ v \in \psi : v = \arg\min_{w \in \psi} |N(w)| \right\} \tag{20}$$

This is called the *Least Unassigned* (LU) rule and allows to obtain much better solutions, although it still produces too many incomplete ones.

Finally a set $\psi_2 \subseteq \psi_1$ is defined by keeping only those vertices in ψ_1 for which, in turn, the total number of free adjacent vertices is minimal, i.e.

$$\psi_2 \triangleq \left\{ v \in \psi_1 : v = \arg\min_{w \in \psi_1} \sum_{u \in N(w)} |N(u)| \right\} \tag{21}$$

This is called the *Least Cumulative Unassigned* (LCU) rule and, thanks to its look-ahead effect, it reduces the chances that a vertex v is left unassigned to any microcontroller since it tends to avoid leaving v isolated. It is convenient to set $\psi = \psi_2$: LCU allows to obtain remarkably better solutions in terms of objective O_1, and partly of O_3 as, in most cases, this mechanism tends to produce chain-like shaped trees, i.e., trees with only two leaves. Objective O_2 is implicitly taken into account by limiting to λ the number of vertices per tree. Fig.3 shows how the LCU rule works. Suppose $T = \{0, 1, 2, 3\}$. Then $N(T) = \{X, Y, Z\}$. All vertices in $N(T)$ have only one free adjacent vertex, except X, so $\psi_1 = \{Y, Z\}$. The cumulative sum of free adjacents for Y is 1, whereas it is 3 for Z , so $\psi_2 = \{Y\}$.

B. Pheromone Structures

The pheromone allows the artificial ants to learn what the attributes of a good solution are. The candidate set must be coherent with the choice. Since in this problem it is necessary to learn how link vertices together, the following possibilities have been tested:

1) Direct Edges (DE): θ is defined as the set of edges linking a candidate vertex and a vertex in current tree, i.e.,

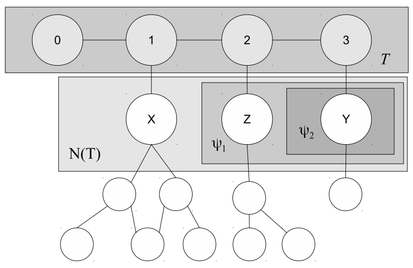


Fig. 3: Candidate set generation.

$$\theta_{\text{DE}} \triangleq \{e \in E, e = (v,i) \vee e = (i,v) : v \in \psi \wedge i \in T\} \quad (22)$$

The pheromone trail is associated with each edge $e \in E$ and is indicated as π_e.

2) Cumulative Edges (CE): in this case

$$\theta_{\text{CE}} = \psi \quad (23)$$

and the value of the corresponding pheromone trail π_v is averaged over all possible edges e linking v to a vertex in T, i.e.,

$$\pi_v \triangleq \frac{\sum_{e \in E_T(v)} \pi_e}{|E_T(v)|} \quad (24)$$

where $E_T(v) \subseteq E$ is defined as:

$$E_T(v) \triangleq \{e \in E, e = (v,i) \vee e = (i,v) : v \in \psi, i \in T\} \quad (25)$$

This is the pheromone structure originally adopted by the authors [1].

3) Direct Pairs (DP): θ is the set of (virtual) edges linking all vertices in T with all vertices in ψ, i.e.,

$$\theta_{\text{DP}} \triangleq \{e \in P_T(v)\} \quad (26)$$

where

$$P_T(v) \triangleq \{e = (i,j), i < j : \quad (27)$$
$$(i \in \psi \wedge j \in T) \vee (i \in T \wedge j \in \psi)\}$$

The pheromone is associated with the same edges.

4) Cumulative Pairs (CP): as in CE but applied to pairs defined in the previous case, i.e.,

$$\theta_{\text{CP}} = \psi \quad (28)$$

and the pheromone trail is averaged over all edges in $P_T(v)$ ending or beginning in v, i.e.,

$$\pi_v \triangleq \frac{\sum_{e \in P_T(v)} \pi_e}{|P_T(v)|} \quad (29)$$

5) Naive Clustering (NC) in this final case, the element to be inserted is a vertex, i.e.,

$$\theta_{\text{NC}} \triangleq \psi \quad (30)$$

and the pheromone π_c is associated to the couple $c = (T, v)$, where T is the tree under construction and $v \in \theta_{\text{NC}}$. This representation is the only one which works also with empty trees, whereas the others require the adoption of dummy edges to bootstrap the construction.

VI. Experimental Analysis

The MSH and the ACO algorithms, the latter with all of the pheromone configurations described above, have been tested on the same instances and with the same termination condition, i.e., a maximum time of 300 seconds. No results of a MIP solver are reported as it has already been shown that the formulation *as-is* is definitely not competitive [1]. Since both algorithms are randomized, 10 independent runs were executed for each instance. All tests were executed on a 2.83 GHz Intel Core 2 Quad CPU Q9550, with 4GB of RAM, running Linux (Ubuntu with 3.2.0-26-generic-pae, 32 bit). Instances are grouped in two datasets: the first contains instances corresponding to patches for some real parts of the *iCub* robot platform[1], which have a number of vertices ranging from 6 up 232; the second contains artificially generated instances with a number of vertices ranging from 35 up to 2470. In the following, the instance associated with real parts from *iCub* are identified by the prefix “i”. These instances correspond respectively to: i1 = Left Upper Arm, lower part, i2 = Left Hip, i3 = Right Upper Forearm, i4 = Torso, i5 = Lower Torso; whereas instances in the synthetic dataset have a prefix “s” followed by the number of vertices. The first four columns of Table I show, for each instance, the number of vertices, microcontrollers and the desired occupancy level λ.

Robot designers require a lexicographic ordering $O_1 \prec O_2 \prec O_3$, where $a \prec b$ means that a has a priority higher than b. In fact, they consider unacceptable (or at least very low quality ones) those solutions that do not assign all the skin elements to the microcontrollers (1), and *microcontroller load balancing* (2) is considered more important than *skin element spreading* (3). To model this priority, the weights in the objective function (5) must be set in order to satisfy $w_1 \gg w_2$ and $w_2 \gg w_3$. In particular the following set of values was considered suitable for the objective weights: $w_1 = 1000$, $w_2 = 10$, $w_3 = 1$. Although all of the algorithms described explicitly enforce the required lexicographic ordering of solutions, this assignment allows the comparison with the solutions obtained with a MIP solver [1].

Table I summarizes the performance of the algorithms: the first column (ID) indicates the instance, the second (BV) the value of the best solution found during the experiments and the remaining columns report the average *Relative Percent Deviation* (RPD) of MSH and of the ACO algorithms with the different pheromone configurations (DE, CE, DP, CP, NC). The last two column report the range between the worst and the best solution found respectively by all algorithms and by ACO algorithms only. Each row is dedicated to a single instance, and the best value is highlighted in boldface. The last two rows show the average and the standard deviation respectively. The RPD is calculated as follows:

$$\text{RPD} = \frac{z - \text{BV}}{\text{BV}} \cdot 100 \quad (31)$$

where z is the value of the objective function found by the algorithm under test.

[1]Please refer to the official iCub website at `www.icub.org`.

TABLE I: Instances characteristics and computational results.

								ACO			range (all)	range (ACO)
ID	$\|V\|$	$\|K\|$	λ	BV	MSH	DE	CE	DP	CP	NC	(max - min)	(max - min)
i1	6	2	3	0.3851	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
i2	9	1	9	0.6792	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
i3	37	4	13	3.1087	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
i4	125	9	16	1.4905	0.023%	0.032%	0.033%	**0.022%**	0.030%	0.035%	0.01	0.01
i5	232	15	16	0.3899	0.240%	0.018%	**0.010%**	0.165%	0.171%	0.215%	0.23	0.21
s35	35	3	12	0.5193	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
s40	40	3	14	0.7179	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
s54	54	4	14	0.5406	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	**0.000%**	0	0
s60	60	4	15	0.1798	0.006%	0.217%	**0.000%**	1.385%	0.063%	0.063%	1.38	1.38
s77	77	5	16	0.5219	0.093%	**0.000%**	0.076%	0.678%	**0.000%**	0.630%	0.68	0.68
s84	84	6	14	0.1230	2.069%	**0.000%**	**0.000%**	**0.000%**	**0.000%**	2.107%	2.11	2.11
s104	104	7	15	0.2027	0.768%	0.320%	0.360%	0.332%	**0.000%**	1.962%	1.96	1.96
s112	112	7	16	0.1063	1.834%	0.327%	1.906%	0.966%	**0.105%**	0.121%	1.8	1.8
s135	135	9	15	0.0840	4.900%	**0.000%**	**0.000%**	4.035%	1.527%	4.336%	4.9	4.34
s170	170	11	16	0.4137	0.482%	0.150%	**0.012%**	0.521%	0.283%	0.271%	0.51	0.51
s198	198	13	16	0.5440	0.263%	0.150%	0.125%	0.079%	**0.052%**	0.216%	0.21	0.16
s252	252	16	16	0.2070	0.730%	0.199%	**0.150%**	0.349%	0.357%	0.372%	0.58	0.22
s322	322	21	16	0.4570	0.201%	0.091%	**0.079%**	0.126%	0.175%	0.178%	0.12	0.1
s432	432	27	16	0.0312	1.970%	**0.133%**	0.511%	0.705%	0.622%	1.152%	1.84	1.02
s608	608	38	16	0.0225	2.230%	**0.309%**	0.914%	1.205%	0.718%	1.128%	1.92	0.9
s874	874	55	16	0.0840	0.324%	**0.041%**	0.113%	0.128%	0.154%	0.135%	0.28	0.11
s1344	1344	84	16	0.0105	1.125%	0.366%	**0.106%**	0.381%	0.665%	0.110%	1.02	0.56
s2470	2470	155	16	0.0461	0.082%	0.018%	**0.013%**	0.030%	0.044%	0.035%	0.07	0.03
				average	0,754%	**0,103%**	0,192%	0,483%	0,216%	0,568%		
				stdev	1,174%	**0,127%**	0,431%	0,876%	0,364%	1,024%		

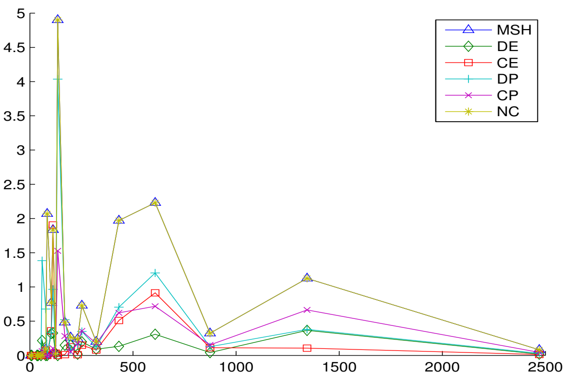


Fig. 4: Algorithms performance.

TABLE II: 95% confidence intervals.

	lo	avg	up	stdev
MSH	0.42%	1.02%	1.62%	1.27%
DE	0.08%	0.14%	0.20%	0.13%
CE	0.03%	0.26%	0.49%	0.49%
DP	0.19%	0.65%	1.11%	0.97%
CP	0.10%	0.29%	0.48%	0.40%
NC	0.23%	0.77%	1.31%	1.13%

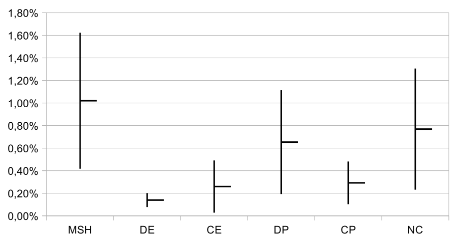


Fig. 5: Average results for MSH and ACOs with 95% confidence intervals.

The same values are visualized in Fig. 4, which plots the results as a function of the number of vertices.

First of all, although not shown, it is remarkable that every algorithm was able to produce a complete solution, i.e., with $O_1 = 0$, in every run on every instance. This is a very satisfying performance on the point of view of the subject matter experts. From the Table it stems out that every algorithm proved to be very efficient on the less challenging instances, namely i1-i3 and s0035 - s0054: not only 0% RPD was achieved on all of them, but they actually constantly kept finding a solution with the same (best) cost in all of the replications. In the graph in Fig. 4, this corresponds to those points, on the left, where all the lines reach the horizontal axis. At this stage of the analysis, it is not possible yet to state that an algorithm was clearly dominated by some other, although the ACO with the Direct Edges pheromone structure achieves the best average results and the tightest standard deviation. Considering the averages the remaining algorithms ranked as follows: CE, CP, DP, NC and MSH; whereas looking a the standard deviation the order was: DE, CP, CE, DP, NC and DE. The right part of the graph reinforces the observation that DE, CE and CP perform better on large scale instances. To gain some more insight, it is useful to filter out the simplest instances and compute the confidence intervals. Table II shows, on the rows, the average RPD (avg) of each algorithm, together with the 95% confidence intervals (lo, up) and the standard deviation (stdev). For convenience, the same intervals are also plotted in Fig. 5.

The confidence interval of DE does not overlap with that associated to MSH, which allows to conclude that the improvement obtained with the Direct Edge pheromone representation is statistically significant. It pays to add this learning mechanism to the purely random constructive mechanism adopted by the MSH. Interestingly, DE is also better than NC, for which the interval is largely coincident with that of MSH, and this is remarkable: apparently, the clustering information associated with the couple $c = (T, v)$, where T is a tree and v is a vertex is not powerful enough. The reason cannot depend on the choice of the edge to be inserted in T because, given vertex v, the contribution to the objective function is univocally defined. To understand one possible reason why NC did not perform well, consider a graph for which the set of vertices can be partitioned into two disjoint subsets V_1 and V_2, and suppose that it is possible to obtain a solution for which V_1 and V_2 are assigned to trees T_1 and T_2 respectively. Then, a completely equivalent solution can be obtained by swapping the assignments, i.e., by assigning V_2 to T_1 and V_1 to T_2. This kind of representation suffers the inefficiency due to its embedded strong symmetry: since equivalent solutions can be obtained simply by performing cyclic permutations of the assignments represented in couples c, the algorithm probably wastes a lot of CPU time to determine which of these equivalent assignments works best. It can be concluded that, at least on the given instances, it is not efficient to represent the clustering information by specifying the tree to which a vertex must be assigned. Instead, remaining structures try to represent the fact that two or more vertices must be grouped together in a good solution. The performance of DE are superior also of those of DP: even if there is a minimal intersection between the two, the interval of DP is almost entirely contained into that of NC. DE, CE and CP look the best configurations. The interval of DE is embedded into that of CE and overlaps for almost its 80% with that of CP, so no statistical significance of the superiority of DE can be inferred.

If a single pheromone had to be selected, then DE would be a good choice, as it is the only that guarantees superior performances with respect to those offered by MSH. CE and CP would be the next best alternative. This ranking allows to speculate that two characteristics of a good pheromone structure are: (i) the use of topology and (ii) the information on the fact that two or more elements (i.e., vertices) must belong to the same cluster. DE and CE have both characteristics, while CP only have the second one. Finally, it is worth noting that on the two largest instances, those with 1344 and 2470 vertices, the situation changes: CE performs best, and NC and DP close the gap. These two final observations will be addressed in future research.

VII. Conclusions

Five different pheromone structures for an Ant Colony Optimization have been designed and tested to solve a minimum cost *Constrained Spanning Forest* problem arising in robotic skin design. These are Direct Edges (DE), Cumulative Edges (DE), Direct Pairs (DP), Cumulative Pairs (DP) and Naive Clustering (NC). The proposed heuristics were tested against real and synthetic instances: results show the effectiveness of all methods but suggest that some structures work better than others. The simplest structure (DE), performs definitely better than the simple randomized restart MSH algorithm obtained disabling the learning capability given by the pheromone and than the Naive Clustering structure. DE is superior also of the CP method but no statistical evidence that DE is better than CE, DP has been found. Moreover, the behaviour of the algorithms seems to be sensitive to the scale of the problem. At the same time, there is a need for automatic methods that can handle an ever increasing number of sensors per area unit, obtained thanks to miniaturization and technology improvements, so even larger cases will be considered in future research.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 231500/ROBOSKIN.

References

[1] D. Anghinolfi, G. Cannata, F. Mastrogiovanni, C. Nattero, and M. Paolucci, "Heuristic approaches for the optimal wiring in large scale robotic skin design," *Computers & Operations Research*, vol. 39, no. 11, pp. 2715 – 2724, 2012.

[2] G. Cannata, M. Maggiali, G. Metta, and G. Sandini, "An embedded artificial skin for humanoid robots," in *Proceedings of the 2008 IEEE International Conference on Multi-sensor Fusion and Integration (MFI'08)*, Seoul, Korea, August 2008.

[3] G. Cannata, R. Dahiya, M. Maggiali, F. Mastrogiovanni, G. Metta, and M. Valle, "Modular skin for humanoid robot systems," in *Proceedings of the 4th International Conference on Cognitive Systems (CogSys'10)*, Zurich, Switzerland, January 2010.

[4] Schmitz A., Maiolino P, Maggiali M., Natale L., Cannata G., and Metta G., "Methods and Technologies for the Implementation of Large Scale Robot Tactile Sensors," *IEEE Trans. on Robotics - Special Issue on Robot Sense of Touch*, vol. 27, no. 3, pp. 389–400, 2011.

[5] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[6] A. Gupta, J. Lafferty, H. Liu, L. Wasserman, and M. Xiu, "Forest density estimation," Carnegie Mellon University, Tech. Rep., 2010.

[7] C. Imielinska, B. Kalantari, and L. Khachiyan, "A greedy heuristic for a minimum-weight forest problem," *Operations Research Letters*, vol. 14, no. 2, pp. 65 – 71, 1993.

[8] J. Monnot and S. Toulouse, "The path partition problem and related problems in bipartite graphs," *Operations Research Letters*, vol. 35, no. 5, pp. 677 – 684, 2007.

[9] C. Bazgan, B. Couëtoux, and Z. Tuza, "Complexity and approximation of the constrained forest problem," *Theoretical Computer Science*, vol. 412, no. 32, pp. 4081 – 4091, 2011.

[10] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *J. ACM*, vol. 7, pp. 326–329, October 1960.

[11] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey." *Theoretical Computer Science*, vol. 344, pp. 243–278, 2005.

[12] D. Anghinolfi and M. Paolucci, "A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem." *International Journal of Operations Research*, vol. 5, pp. 1–17, 2008.

[13] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem." *IEEE Trans. on Evolutionary Computation*, vol. 1, pp. 53–66, 1997.

[14] T. Stützle and H. H. Hoos, "Max-min ant system." *Future Generation Computer System*, vol. 16, pp. 889–914, 2000.