Autonomous deployment of self-organizing mobile sensors for a complete coverage

N. Bartolini, T. Calamoneri, E. G. Fusco, A. Massini, S. Silvestri

Department of Computer Science, Sapienza University of Rome, Italy
{bartolini, calamo, fusco, massini, simone.silvestri}@di.uniroma1.it

Abstract. In this paper we propose an algorithm for the autonomous deployment of mobile sensors over critical target areas where sensors cannot be deployed manually. The application of our approach does not require prior knowledge of the working scenario nor any manual tuning of key parameters. Our algorithm is completely distributed and sensors make movement decisions on the basis of locally available information. We prove that our approach guarantees a complete coverage, provided that a sufficient number of sensors are available. Furthermore, we demonstrate that the algorithm execution always terminates preventing movement oscillations. We compare our proposal with one of the most acknowledged algorithms by means of extensive simulations, showing that our algorithm reaches a complete and more uniform coverage under a wide range of operating conditions.

1 Introduction

Research in the field of mobile wireless sensor networks is motivated by the need to monitor critical scenarios such as wild fires, disaster areas, toxic regions or battlefields, where static sensor deployment cannot be performed manually. In these working situations, sensors may be dropped from an aircraft or sent from a safe location.

We address the problem of coordinating sensor movements to improve the coverage of an Area of Interest (AoI) with respect to the initial deployment, which typically is neither complete nor uniform. Centralized solutions are inefficient because they require either a prior assignment of sensors to positions or a starting topology that ensures the connectivity of all sensors (for global coordination purposes). Therefore, feasible and scalable solutions should employ a distributed scheme according to which sensors make local decisions to meet global objectives. Due to the limited power availability at each sensor, energy consumption is a primary issue in the design of any self-deployment scheme for mobile sensors. Since sensor movements and, to a minor extent, message exchanges, are energy consuming activities, a deployment algorithm should minimize movements and message exchanges during deployment.

Among the previous proposals, the virtual force approach (VFA) [1–4] models the interactions among sensors as a combination of attractive and repulsive forces. Other approaches are inspired by the physics as well: in [5] the sensors

are modelled as particles of a compressible fluid, in [6] the theory of gas is used to model sensor movements in the presence of obstacles. A similar approach is used in [7] to give a unified solution to the problem of deployment and dynamic relocation of mobile sensors in an open environment.

Other proposals are inspired by computational geometry. Among these, the Voronoi approach (VOR) [8] provides general rules for sensor movements on the basis of a local calculation of the Voronoi diagrams determined by the sensor deployment. A variant of this approach [9] analyzes the problem of sensor deployment in a hybrid scenario, with both mobile and fixed sensors in the same environment. The authors of [10] propose the use of Delaunay triangulation techniques to obtain a regular tessellation of the AoI. In [11] a novel approach to sensor deployment is designed with particular emphasis on operative settings where coverage does not imply connectivity.

Differently from our work, the cited approaches require the manual tuning of constants and thresholds whose proper values are closely dependent on the particular operative setting.

We propose an original algorithm for mobile sensor deployment, Push & Pull. It does not require any prior knowledge of the operative scenario nor any manual tuning of key parameters. It constitutes a wide extension of our previous proposal, Snap & Spread [12] to which we added two basic activities to guarantee the coverage completeness and uniformity and to improve the network fault tolerance. Push & Pull is based on the interleaved execution of four activities designed to produce a hexagonal tiling by moving sensors out of high density regions and attracting them towards coverage holes. Decisions regarding the behavior of each sensor are based on locally available information.

Our algorithm has the basic self-* properties of autonomic computing, i.e. self-configuration and self-adaptation. Its design follows the grassroots approach [13] to autonomic computing. This way self-organization emerges without the need of external coordination or explicit control, giving rise to a fully decentralized algorithm, according to which the sensor behavior is democratic and peer structured. We formally prove that our algorithm reaches a final stable deployment and a complete coverage of the AoI in a finite time. We ran extensive simulations to evaluate the performance of our algorithm and compare it to existing solutions. Experimental results show that Push & Pull performs better than one of the most acknowledged and cited algorithms [8].

2 The Push & Pull algorithm

Let V be a set of equally equipped sensors with isotropic communication and sensing capabilities. The transmission radius is $R_{\mathtt{tx}}$ and the sensing radius is $R_{\mathtt{s}}$. We propose a distributed algorithm according to which sensors aim at forming a hexagonal tiling over the AoI, with hexagon side length equal to $R_{\mathtt{s}}$. This setting guarantees both coverage and connectivity when $R_{\mathtt{tx}} \geq \sqrt{3}R_{\mathtt{s}}$ and is not restrictive as most wireless devices can adjust their transmission range by properly setting their transmission power. Finally, we assume that sensors are

endowed with location capabilities. It should be noted that this assumption is only necessary if the algorithm has to be applied over a specific AoI, with given coordinates. As in [7] this assumption can be removed when dealing with an open environment.

A sensor being positioned in the center of a hexagon is referred to as a *snapped sensor*. Given a sensor x, snapped to the center of a hexagon, we define *slaves of* x all the other sensors lying in its hexagon. We denote by $S(x)$ the set of slaves of x and by $Hex(x)$ the hexagonal region whose center is covered by x. All sensors that are neither snapped nor slaves are called *free*. We define $L(x)$, the set composed by the free sensors located in radio proximity to x and by its slaves $S(x)$.

The algorithm PUSH & PULL is based on the interleaved execution of four fundamental activities: *Snap*, *Push*, *Pull* and *Merge*. The coordination of these activities requires the definition of a communication protocol that we do not detail due to space limitations. The interested reader may find more information in [14].

Snap activity

At the beginning, each sensor may act as starter of a snap activity from its initial location at an instant randomly chosen over a given time interval $T_{\texttt{start}}$. Initially, several sensors may likely create different tiling portions. A starter sensor elects its position as the center of the first hexagon of its portion. It selects at most six sensors among those located within its transmission radius and makes them snap to the center of adjacent hexagons. Such snapped sensors, in their turn, give start to analogous activities, thus expanding the boundary of their tiling portion.

More precisely, a snapped sensor x performs a *neighbor discovery*, that allows x to gather information regarding $S(x)$ and all the free and snapped sensors located in radio proximity. After the neighbor discovery, x determines whether some adjacent snapping positions are still to be covered and leads the corresponding snap activity. The sensor x snaps the closest sensor in $L(x)$ to each uncovered position. A snapped sensor leads the snapping of as many adjacent hexagons as possible. If all the hexagons adjacent to $Hex(x)$ have been covered, x stops any further snapping and gives start to the push activity. Otherwise, if some hexagons are left uncovered because no more sensors in $L(x)$ are available, x starts the pull activity.

Push activity

After the completion of the snap activity, a snapped sensor x, may still have some slave sensors inside its hexagon, so $S(x) \neq \emptyset$. In this case, it can proactively push such slave sensors towards lower density areas.

Given two snapped sensors x and y located in radio proximity from each other, x may offer one of its slaves to y and push it inside the hexagon of y if $|S(x)| \geq |S(y)|+1$. Notice that, when $|S(x)| = |S(y)|+1$ the flow of a sensor from $Hex(x)$ to $Hex(y)$ leads to a symmetric situation in which $|S(x)| + 1 = |S(y)|$,

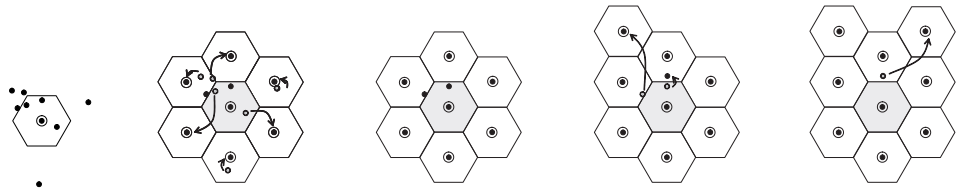


Fig. 1. Snap and Push: an example.

leading to possible endless cycles. In such cases we restrict the push activity to only one direction: x pushes its slave to y only if $id(y) < id(x)$, where $id(\cdot)$ is any function such that $id : V \to \mathbb{N}$.

We formalize these observations by defining the following condition, that enables the movement of a sensor from $Hex(x)$ to $Hex(y)$:
Moving Condition: $\{|S(x)| > |S(y)| + 1\} \ \vee \ \{|S(x)| = |S(y)| + 1 \ \wedge \ id(x) > id(y)\}$.
The snapped sensor x pushes one of its redundant sensors towards the hexagon of the snapped sensor y which has the lowest number of slaves among those in radio connectivity with x. If more than one hexagon contains the minimum number of sensors, the closest to x is preferred. Among its slaves, x selects the sensor to push according to the criterion of minimum traversed distance to $Hex(y)$.

Figure 1 shows an example of the execution of the first two activities. Figure 1(a) depicts the initial configuration, with nine randomly placed sensors and highlights the starter sensor $s_{\texttt{init}}$ creating the first hexagon of the tiling. In Figure 1(b) the starter sensor $s_{\texttt{init}}$ selects six sensors to snap in the adjacent hexagons, according to the minimum distance criterion. Figure 1(c) shows the configuration after the snap activity of $s_{\texttt{init}}$. In Figure 1(d), a deployed sensor starts a new snap activity while $s_{\texttt{init}}$ starts the push activity sending a slave sensor to a lower density hexagon. In Figure 1(e) the deployed sensor snaps the sensor just received from the starter, reaching the final configuration.

As a consequence of the push activity, slave sensors generally consume more energy than snapped sensors, because they are involved in a larger number of message exchanges and movements. Thus we let slave and snapped sensors occasionally exchange their role in order to balance the energy consumption. Any time a slave s has to make a movement across a hexagon occupied by the snapped sensor z, the two sensors perform a role exchange if the residual energy of s is lower than the one of z.

Pull activity

The sole snap and push activities are not sufficient to ensure the maximum expansion of the tiling, and may likely leave coverage holes. Even when the number of available sensors is sufficient to completely cover the AoI, a snapped

sensor x could not have any sensor in $L(x)$) to cover the adjacent vacant snapping positions. This may happen due to the Moving Condition introduced to avoid moving cycles. For this reason, we introduce the pull activity: snapped sensors detecting a coverage hole adjacent to their hexagons, and not having available sensors to snap, send hole trigger messages in order to attract slave sensors and make them fill the hole.

If a snapped sensor x, with $L(x) = \emptyset$, detects a hole, and the Moving Condition is not verified for any of its snapped neighbors, then the following trigger mechanism is activated. The sensor x temporarily alters the value of its id function to 0 and notifies its neighbors of this change. Then x waits until either a new slave comes into its hexagon or a timeout occurs. If a new slave enters in $Hex(x)$, x sets back its id value and snaps the new sensor, filling the hole. If otherwise the timeout expires and the hole is still present, the trigger mechanism is extended to the adjacent hexagons of x, whose snapped sensors set their id value to 1 and notify their neighbors.

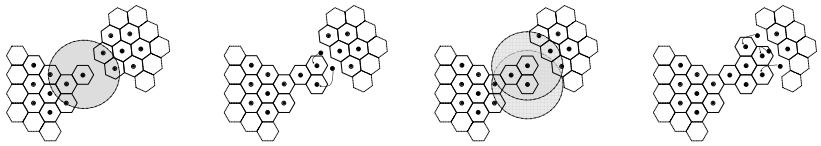
Each snapped sensor involved in the trigger extension mechanism sets its id to a value that is proportional to the distance from x. All the timeouts related to each new extension are set in proportion to the maximum distance reached by the trigger mechanism. This mechanism is iterated by x over snapped sensors at larger and larger distance in the tiling until the hole is covered. At this point, as a consequence of timeouts, each involved node sets back its id to the original value.

Observe that, more snapped sensors adjacent to the same hole may independently activate the trigger mechanism. In this case, the only message with lowest id is honored. The detection of several holes may cause the same node to receive more a number of trigger messages that are stored in a pre-emptive priority queue, giving precedence to the messages related to the closest hole.

Tiling merge activity

The fact that many sensors act as starters implies the possible creation of several tiling portions with different orientations. The tiling merge activity starts when two tiling portions come in radio proximity. We propose a merge mechanism according to which as soon as a sensor x receives a neighbor discovery message from another tiling portion, it chooses to belong to the oldest one. The sensor x discriminates this situation by evaluating the time-stamp of the starter action that is propagated at each snap action.

Figure 2 shows an example of the execution of the tiling merge activity. Figure 2(a) shows two tiling portions meeting each other. The portion on the left has the oldest time-stamp, hence it absorbs the other one. Two nodes of the right portion detect the presence of an older tiling and abandon their original tiling to honor snap commands coming from a sensor of the left portion (Figure 2(b)). These just snapped nodes, now belonging to the older portion, detect the presence of three nodes belonging to the right one (Figure 2(c)) and snap them as soon as they leave their portion (Figure 2(d)).



(a) (b) (c) (d)

Fig. 2. Tiling merge activity: an example.

3 Algorithm properties

In this section we discuss some key properties of the PUSH & PULL algorithm: coverage, connectivity and termination.

3.1 Coverage and connectivity

We denote by $N_{\texttt{tight}}$ the *tight number of sensors*, that is the maximum number of hexagons necessary to cover the AoI for each possible initial position of the sensor set and each possible tiling orientation. Notice that an upper bound on this number can be calculated by increasing the AoI with a border whose width is the maximal diameter of the tiling hexagon and dividing this increased area AoI' by the area of the hexagon. Formally, $N_{\texttt{tight}} \leq \lceil \frac{\text{Area(AoI')}}{\text{Area(Hex)}} \rceil$. This upper bound is valid regardless of the number of tiling portions generated by different starter sensors.

Theorem 1. *Algorithm* PUSH & PULL *guarantees a complete coverage, provided that at least the tight number of sensors are available.*

Proof. Let us assume that a coverage hole exists and let x be the sensor which detects it. The hypothesis on the number of sensors implies that it certainly exists a hexagon with at least one redundant slave. Let us call C_x the connected component containing the sensor x. Two different cases may occur depending on the position of redundant slaves with respect to C_x:

- A redundant slave exists in C_x.
  The snapped sensor x starts the trigger mechanism that eventually reaches a hexagon with a redundant slave. Such a slave is then moved towards x and finally fills the hole.
- All redundant slaves belong to connected components separated from C_x.
  The area surrounding each connected component is in fact a coverage hole that will eventually be detected by a snapped node located at the boundary. According to the previous item, all the separated connected components containing redundant slaves will expand themselves to fill as many coverage

holes as possible. Since by the hypothesis the number of sensors is at least $N_{\texttt{tight}}$, it certainly exists a component containing redundant slaves that will eventually merge with C_x, leading to the situation described in the first item, thus proving the theorem.

We assume that sensors operate with $R_{\texttt{tx}} \geq \sqrt{3}R_{\texttt{s}}$. Simple geometrical considerations allow us to conclude that under this assumption, a hexagonal tiling with side $R_{\texttt{s}}$, i.e. where the distance between any two grid neighbors is $\sqrt{3}R_{\texttt{s}}$, guarantees minimal node density (as argued in [15]) and connectivity.

3.2 Termination of Push & Pull

Let $L = \{\ell_1, \ell_2, \ldots, \ell_{|L|}\}$ be the set of snapped sensors.

Definition 1. *A* network state *is a vector* $\mathbf{s} =< s_1, s_2, \ldots, s_{|L|} >$*, where* $s_i = |S(i)| + 1$*, is the number of sensors deployed inside the hexagon* $Hex(i)$*,* $\forall i = 1, \ldots, |L|$.

Definition 2. *A state* $\mathbf{s}$ *is* stable *if the Moving Condition is false for each couple of snapped sensors located in radio proximity to each other.*

Theorem 2. *Algorithm* Push & Pull *terminates in a finite time.*

Proof. Due to Theorem 1, the expansion of the tiling generated by Push & Pull eventually ends either because all sensors have been snapped or the AoI has been completely covered by snapped sensors. In the first case, no algorithm actions are necessary, then the algorithm terminates producing a stable state of the network. Thus, in order to prove the theorem, it suffices to prove that, once the AoI is fully covered by snapped sensors, the algorithm produces a stable network state in a finite time. After the complete coverage of the AoI, the set L of snapped sensors remains fixed. The value of the order function related to each snapped sensor, $id(\ell_i)$ can be modified by the pull activity only a finite number of times and remains steady onward.
Let us define $f : \mathbb{N}^{|L|} \to \mathbb{N}^2$ as follows: $f(\mathbf{s}) = \left(\sum_{i=1}^{|L|} s_i^2, \sum_{i=1}^{|L|} id(\ell_i)s_i\right)$.
We say that $f(\mathbf{s}) \succ f(\mathbf{s}')$ if $f(\mathbf{s})$ and $f(\mathbf{s}')$ are in lexicographic order. Observe that function f is lower bounded by the pair $(|L|, \sum_{i=1}^{|L|} id(\ell_i))$, in fact $1 \leq s_i \leq |V|$. Therefore, if we prove that the value of f decreases at every state change from $\mathbf{s}$ to $\mathbf{s}'$, we also prove that no infinite sequence of state changes is possible.

Let us consider a generic state change which involves the snapped sensors x and y, with x sending a slave sensor to $Hex(y)$. We have that $s_i = s'_i \quad \forall i \neq x, y$, and $s'_x = s_x - 1$ and $s'_y = s_y + 1$. As the transfer of the slave has been done according to the Moving Condition, two cases are possible: either $s_x > s_y + 1$, or $(s_x = s_y + 1) \wedge (id(x) > id(y))$. In the first case, $s_x > s_y + 1$ trivially implies that $\sum_{i=1}^{|L|} s_i^2 > \sum_{i=1}^{|L|} {s'}_i^2$. In the second case, from $s_x = s_y + 1$ and $id(x) > id(y)$, easy calculations imply that $\sum_{i=1}^{|L|} s_i^2 = \sum_{i=1}^{|L|} {s'}_i^2$ and $\sum_{i=1}^{|L|} id(\ell_i)s_i > \sum_{i=1}^{|L|} id(\ell_i){s'}_i$. Therefore in both cases $f(\mathbf{s}) \succ f(\mathbf{s}')$. The function f is lower bounded and always decreasing of discrete quantities (integer values) at any state change. Thus, after a finite time the network will be in a *stable state*, thus the theorem is proved.

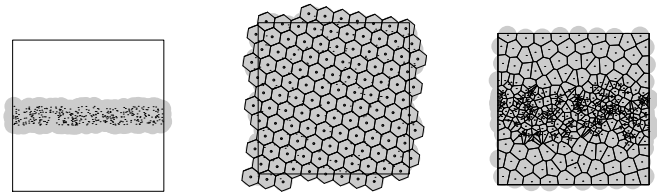
4 Simulation results

In order to evaluate the performance of PUSH & PULL and to compare it with previous solutions, we developed a simulator on the basis of the wireless module of the OPNET modeler software [16]. We compared our proposal to one of the most acknowledged and cited algorithms [8], which is based on the use of Voronoi diagrams. According to this approach, each sensor adjusts its position on the basis of a local calculation of the Voronoi cell determined by the current sensor deployment. This information is used to detect coverage holes and consequently calculate new target locations according to three possible variants. Among these variants we chose MiniMax, that is the one that gives better guarantees in terms of coverage extension. We also adopted all the mechanisms provided in [8] to preserve connectivity, to guarantee the algorithm termination, to avoid oscillations and to deal with position clustering. In the rest of this section this algorithm will be named VOR.

The experimental activity required the definition of some setup parameters: $R_{\mathtt{tx}} = 11$ m and $R_{\mathtt{s}} = 5$ m. This setting does not significantly affect the qualitative evaluation of PUSH & PULL but is motivated by the need to satisfy the requirement $R_{\mathtt{tx}} \geq 2R_{\mathtt{s}}$ given in [8]. The sensor speed is 1 m/sec. The energy spent by sensors for communications and movements is expressed in energy units (i.e. the cost of receiving one message): a single transmission costs the same as 7 receptions [17], a 1 meter movement costs the same as 300 transmissions [8] and a starting/braking action costs the same as 1 meter movement [8].

The length of the time interval $T_{\mathtt{start}}$ is set to $R_{\mathtt{tx}}/v$, where v is the sensor movement speed. The setting of this parameter ensures that different grid portions are not created too close to each other. Nevertheless, it does not affect the algorithm performance significantly.

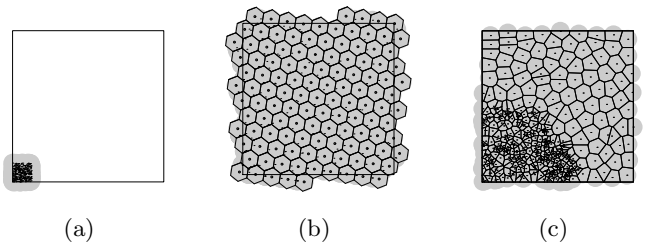
Before showing the performance of our algorithm with respect to VOR, we show some examples of final deployments provided by the two approaches.



(a) (b) (c)

Fig. 3. Comparison between PUSH & PULL and VOR - Trail initial deployment

Figures 3 and 4 show the sensor deployment over a 80 m × 80 m squared AoI. The two initial deployments reflect the realistic scenarios in which sensors



(a) (b) (c)

Fig. 4. Comparison between PUSH & PULL and VOR - Safe-location initial deployment

are dropped from an aircraft (Figure 3(a)) and sent from a safe location at the boundaries of the AoI (Figure 4(a)). For both figures, subfigure (a) represents the initial deployment, while subfigures (b) and (c) show the final deployment obtained by PUSH & PULL and VOR respectively.

We now show the performance of our algorithm with respect to VOR, starting from the initial deployments described above. We studied the algorithm behavior by varying the number of available sensors. In order to give a reliable performance comparison, we show the average results of 30 simulation runs conducted by varying the seed for the generation of the initial random deployment of the sensors.

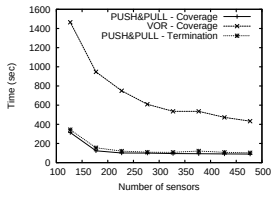


Fig. 5. Term. and coverage time

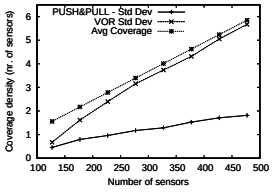


Fig. 6. Coverage density

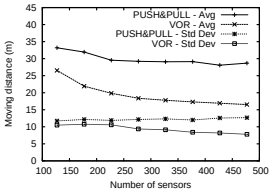


Fig. 7. Traversed distance

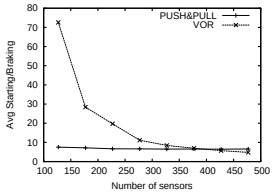


Fig. 8. Starting and braking

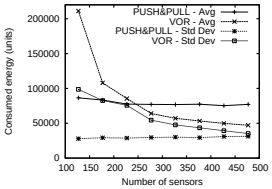


Fig. 9. Energy consumption

Figures 5 through 9 show the performance results for the first set of experiments regarding the trail initial deployment.

Figure 5 shows the coverage and termination time for both the Push & Pull and the VOR algorithms. Notice that, when the number of sensors is close to the tight value, defined in Section 3.1, VOR requires a very long time to achieve a complete coverage, while Push & Pull terminates much earlier. When the number of sensors increases, both algorithms terminate faster, but VOR always requires more time than Push & Pull to achieve its final coverage. Note also that while for the VOR algorithm the termination and coverage completion times coincide, because it moves sensors with the only objective to increase coverage, for Push & Pull some more movements still occur even after the coverage completion. These movements are performed to keep on uniforming the sensor density.

In order to evaluate the coverage uniformity, we compute the coverage density as the number of sensors covering the points of a squared mesh with side 1 m. Figure 6 shows the standard deviation of the coverage density. The standard deviation of the coverage density obtained by Push & Pull is much smaller pointing out a more uniform sensor placement. This result is particularly important as a uniform sensor redundancy is necessary to guarantee fault tolerance and to prolong the network lifetime with selective sensor activation schemes.

Figure 7 shows the average and standard deviation of the distance traversed by the sensors. Push & Pull let sensors move more than VOR because it aims at making the coverage as uniform as possible. Notice that both the average and the standard deviation of the traversed distance of VOR are decreasing with the number of sensors since more and more sensors maintain their initial positions. It should be noted that the result of this comparison must not be interpreted as a negative aspect of our protocol. Indeed, Push & Pull keeps on moving sensors until a quite uniform coverage is reached while the movements determined by VOR terminate as soon as the AoI is completely covered. Hence the average and standard deviation of the traversed distance are more stable under Push & Pull than under VOR when the number of sensors varies.

Figure 8 highlights that VOR spends much more energy than Push & Pull in starting/braking actions. The average value of such energy cost decreases with a growing number of available sensors as the majority of them do not move at all under VOR.

In Figure 9 we give a global evaluation of the above contributions, and show the average and standard deviation of the total consumed energy (i.e. the sum of the contributions due to movements, starting/braking and communications). This figure shows that when the number of sensors is small (lower than about 250 for this experimental setting), although VOR consumes less energy in movements, the impact of starting/braking actions is not negligible and compensate the higher cost of movements paid by Push & Pull. When the number of sensors grows, VOR consumes less energy with respect to Push & Pull as a large part of the sensors are left unmoved.

In all the simulated situations, the execution of VOR implies that a number of sensors are moved away from overcrowded regions toward uncovered areas. As soon as all the coverage holes are eliminated, VOR stops, leaving some zones

with very low density coverage. Such zones represent possible points of future failures and coverage holes. PUSH & PULL mitigates this problem by sending much more sensors than VOR to the farthest and less dense regions of the AoI.

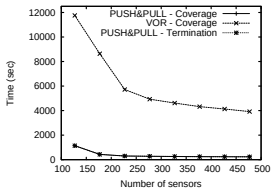


Fig. 10. Term. and coverage time

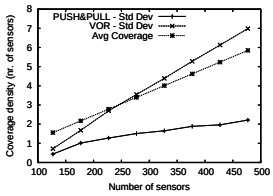


Fig. 11. Coverage density

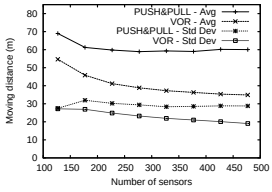


Fig. 12. Traversed distance

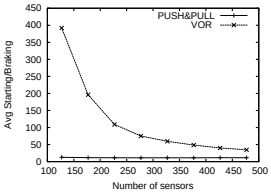


Fig. 13. Starting and braking

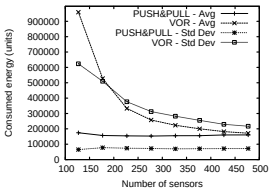


Fig. 14. Energy consumption

In the second set of experiments the initial deployment consists in a high density region at the boundaries of the AoI as depicted in Figure 4(a). It is worth noting that this initial deployment constitutes a critical scenario for VOR as this algorithm works at its best for more uniform initial sensor distributions. Indeed, Figure 10 shows that VOR requires much more time than in the previous set of experiments to achieve its final deployment. This figure also shows that VOR is much slower than PUSH & PULL in completing the coverage of the AoI. For what concerns the density of the distribution, Figure 11 shows that even in this operative setting, VOR terminates as soon as the AoI is completely covered, without uniforming the density of the sensor deployment. This implies that VOR spends less energy in movements (see Figure 12) than PUSH & PULL but at the expense of the quality of the final coverage in terms of uniformity. Furthermore, VOR shows a much higher number of starting/braking actions (see Figure 13) than PUSH & PULL, with a much higher value of the total consumed energy (see Figure 14).

5 Conclusions and future work

We proposed an original algorithm for mobile sensor self-deployment named PUSH & PULL. According to our proposal, sensors autonomously coordinate their movements in order to achieve a complete coverage with moderate energy consumption. The execution of PUSH & PULL does not require any prior knowledge of the operating conditions nor any manual tuning of key parameters as

sensors adjust their positions on the basis of locally available information. The proposed algorithm guarantees the achievement of a complete and stable final coverage, provided that there is a sufficient number of sensors. Some improvements are being considered as a future extension of this work. In particular, it seems reasonable that the algorithm can be generalized in order to guarantee a k-coverage. Mechanisms for obstacle detection and avoidance are also being investigated.

References

1. Zou, Y., Chakrabarty, K.: Sensor deployment and target localization based on virtual forces. Proc. IEEE INFOCOM '03 (2003)
2. Heo, N., Varshney, P.: Energy-efficient deployment of intelligent mobile sensor networks. IEEE Transactions on Systems, Man and Cybernetics **35** (2005)
3. Chen, J., Li, S., Sun, Y.: Novel deployment schemes for mobile sensor networks. Sensors **7** (2007)
4. Poduri, S., Sukhatme, G.S.: Constrained coverage for mobile sensor networks. Proc. of IEEE Int'l Conf. on Robotics and Automation (ICRA '04) (2004)
5. Pac, M.R., Erkmen, A.M., Erkmen, I.: Scalable self-deployment of mobile sensor networks; a fluid dynamics approach. Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS '06) (2006)
6. Kerr, W., Spears, D., Spears, W., Thayer, D.: Two formal fluid models for multi-agent sweeping and obstacle avoidance. Proc. of AAMAS (2004)
7. Garetto, M., Gribaudo, M., Chiasserini, C.F., Leonardi, E.: A distributed sensor relocation scheme for environmental control. The Fourth ACM/IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems, (MASS) (2007)
8. Wang, G., Cao, G., La Porta, T.: Movement-assisted sensor deployment. IEEE Transaction on Mobile Computing **6** (2006)
9. Wang, G., Cao, G., La Porta, T.: Proxy-based sensor deployment for mobile sensor networks. IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS) (2004)
10. Ma, M., Yang, Y.: Adaptive triangular deployment algorithm for unattended mobile sensor networks. IEEE Transactions on Computers **56** (2007)
11. Tan, G., Jarvis, S.A., Kermarrec, A.M.: Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks. Proc. of ICDCS (2008)
12. Bartolini, N., Calamoneri, T., Fusco, E.G., Massini, A., Silvestri, S.: Snap & spread: a self-deployment algorithm for mobile sensor networks. Proc. of IEEE/ACM Int. Conf. on Distributed Computing in Sensor Systems (DCOSS) **3** (2008)
13. Babaoglu, O., Jelasity, M., Montresor, A.: Grassroots approach to self-management in large-scale distributed systems. Unconventional Programming Paradigms. Lecture Notes in Computer Science, Springer Verlag **3566** (2005)
14. Bartolini, N., Massini, A., Silvestri, S.: P&p protocol: local coordination of mobile sensors for self-deployment. http://arxiv.org/abs/0805.1981 (2008)
15. Brass, P.: Bounds on coverage and target detection capabilities for models of networks of mobile sensors. ACM Transactions on Sensor Networks **3** (2007)
16. http://www.opnet.com: Opnet technologies inc.
17. http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/: Smart dust