# Analysis and Comparison of Truly Distributed Solvers for Linear Least Squares Problems on Wireless Sensor Networks

Karl E. Prikopa, Hana Straková, and Wilfried N. Gansterer

University of Vienna, Vienna, Austria
Faculty of Computer Science

**Abstract.** The solution of linear least squares problems across large loosely connected distributed networks (such as wireless sensor networks) requires distributed algorithms which ideally need very little or no coordination between the nodes. We first provide an extensive overview of distributed least squares solvers appearing in the literature and classify them according to their communication patterns. We are particularly interested in *truly distributed* algorithms which do not require a fusion centre, cluster heads or any multi-hop communication. Beyond existing methods, we propose the novel least squares solver PSDLS, which utilises a recently developed distributed QR factorisation algorithm. All communication between nodes is exclusively performed within the push-sum algorithm for distributed aggregation.

We analytically compare the communication cost of PSDLS and the existing truly distributed algorithms. In all these algorithms, the communication cost of reaching a predefined accuracy depends on many factors, including network topology, problem size, and settings of algorithm-specific parameters. We illustrate with simulation experiments that our novel PSDLS solver requires significantly fewer messages per node than the previously existing methods to reach a predefined solution accuracy.

## 1 Introduction

We consider the problem of solving the linear least squares problem

$$\min_x \|b - Ax\|_2 \tag{1}$$

for $x \in \mathbb{R}^m$ in a truly distributed way, where $A \in \mathbb{R}^{n\times m}$ with $n \geq m$ and $b \in \mathbb{R}^n$. We are interested in solving such problems over a loosely connected, decentralised network, e. g. a wireless sensor network (WSN), where each node holds part of the input data. In particular, we assume that $A$ is distributed row-wise over the $N$ nodes of the network and that the element $b(i)$ resides on the same node as the $i^{\text{th}}$ row of $A$. For $n > N$, each node contains a block of consecutive rows of $A$.

Many applications in WSNs require the distributed solution of a linear least squares problem, e. g., the reconstruction of physical fields [1], target tracking [2], the solution of the seismic tomography inversion problem [3] when monitoring volcanic activity or localisation [4]. WSNs typically consist of a large number of inexpensive sensor nodes which act autonomously but cooperate with each other to achieve a common goal. Working in a fully decentralised manner allows for decisions to be made on any node. In combination with actuators, the nodes can take autonomous actions in the physical world. Asynchronous communication is an important challenge to be considered in the design of a truly distributed algorithm. The sensor nodes are normally constrained in terms of their resources, primarily their energy supply and computation capabilities. One of the sources of high power consumption is communication. The energy required by the nodes to communicate with other nodes is directly proportional to the communication range. This implies that communicating with the immediate neighbourhood of a node is significantly cheaper than communicating with very distant nodes. Preserving energy also increases the lifespan of the nodes and in turn of the entire network.

As we will summarise in the following section, many distributed least squares solvers can be found in the literature, but most of them do not operate in a truly distributed manner without the need for centralised fusion centres, cluster heads or multi-hop communication. Multi-hop communication requires routing tables, and setting those up requires additional communication. The overhead is particularly large if the routing tables have to be updated frequently.

Dynamic changes and distributed fault tolerance are also important factors in the design of a distributed algorithm for WSNs. Although such difficult scenarios are beyond the scope of this paper, they can be implicitly considered by the use of gossip algorithms for aggregation. The push-sum algorithm [5] used by the PSDLS algorithm proposed in this paper can be directly replaced by fault-tolerant alternatives which are able to recover from silent message loss and temporary or permanent link failures [6,7].

*Synopsis.* In Section 2, we provide an extensive review of the existing literature about distributed least squares solvers and classify them based on their communication patterns. In Section 3, we introduce the new push-sum-based distributed least squares solver PSDLS. Section 4 provides an analytical comparison of the communication cost of PSDLS and the truly distributed algorithms appearing in the literature up until now. Simulation results are presented in Section 5, and our conclusions are summarised in Section 6.

## 2 Existing Distributed Least Squares Solver

In this section, we summarise the efforts presented in the literature for solving the linear least squares problem (1) in a distributed setting. We categorise existing algorithms into three groups: ($i$) *Centralised approaches* using a fusion centre or approaches which require *global communication*, ($ii$) *clustered approaches* where the communication of each node is limited to a subset of the network (cluster) with a cluster head, and (*iii*) *truly distributed approaches* where the communication of each node is limited to its immediate neighbourhood *without* using any multi-hop communication.

## 2.1 Centralised Approaches or Global Communication

A strategy that has been studied extensively is the use of a central unit (*fusion centre*) which performs the computation for the entire network. The fusion centre approach first collects the data from all nodes in the network (global communication), then solves problem (1) at the fusion centre and finally distributes the result to all nodes (global communication). The positioning of the fusion centre is crucial for communication cost and scalability (cf. [3]). There are several drawbacks to this approach: Potential congestion effects (particularly around the fusion centre [8]) can lead to delays and in the worst case to data loss. Multi-hop communication and setting up routing tables incur additional overhead. Last, but not least, the fusion centre becomes a single point of failure. Research on fusion centre approaches often focusses on the efficient accumulation of the data at the fusion centre (see, e. g., [9]). Other efforts perform only parts of the computation at the fusion centre and offload other parts onto the individual nodes (see, e. g., [4]). However, these approaches still require global (multi-hop) communication of each node with the fusion centre.

Reichenbach et al. [4] consider the problem that each node needs to determine its location and analyse three methods for solving the least squares problem arising in this context: normal equations, QR factorisation and singular value decomposition. For all three methods, they split the computation into two parts in order to distribute them between a high performance base station and wireless sensor nodes. The base station computes the computationally intensive tasks and then sends the result to the nodes, which only have to perform low complexity computations to determine their location. This approach significantly reduces the amount of computation performed on the sensor nodes, saving more than 47% of floating-point operations for normal equations and more than 99% for the QR factorisation and the SVD. The disadvantage is the communication cost incurred by the nodes having to send their measurements to the fusion centre either over long distances or with multi-hop communication and non-static routing.

One example for exploiting a specific routing structure is presented by Borgne et al. [9], where the measured data is aggregated at each node towards the fusion centre along a routing tree. The authors extend the basic set of available aggregation functions (minimum, maximum, sum, count and average) to a regression operator which uses the sensor node measurements as input, reducing the amount of data based on the regression model. The advantage of this approach is the reduction of the communication range of the nodes to a localised neighbourhood. However, the final result is only available at the fusion centre, which in the event of a failure leads to the breakdown of the entire computation.

The distributed multisplitting method [10], based on the parallel multiplitting method by Reanut [11], applies the well-known fixed-point iteration methods Jacobi, Gauss-Seidel and successive over-relaxation to the normal equations.

The matrix $A$ is distributed column-wise over the nodes and weighting matrices are used to recombine the solutions of the local problems, which are independent problems resulting from the linear multisplitting of $A$. Note that in this method, the solution $x$ is not replicated, but distributed across the nodes. In each iteration a vector of size $n$ has to be broadcast to all other nodes (global communication).

The distributed modified conjugate gradient least squares (D-MCGLS) algorithm [10] exploits the fact that the conjugate gradient method can be applied to the symmetric and positive definite normal equations. It is also based on a parallel method, MCGLS by Yang and Brent [12], which is targeted at distributed memory architectures. Yang and Brent have improved the parallel performance of the standard CGLS method by reducing the global synchronisation points for the inner products. D-MCGLS requires $A$ to be distributed row-wise. If $A$ is not symmetric, for each local row of $A$, the node also needs to have the corresponding column locally. Each node has to use the same initialisation for $x$. In each iteration, a vector of length $m$ and a scalar value have to be broadcast to all other nodes in the network (global communication).

## 2.2 Clustered Approaches

A first step towards a more decentralised setting than the fusion centre approaches summarised in Section 2.1 is based on clustering. The network is divided into clusters. In each cluster, one node acts as the cluster head, which often is more powerful than the other nodes in the cluster. The division is based on a certain criterion, e. g., on the geographical location of the nodes or on the predefined communication radius of the cluster head. The cluster heads act as intermediate fusion centres for the clusters. The nodes of a cluster only communicate with their cluster head and with nodes within the same cluster. Compared to the fusion centre approaches, a multi-tier model is used where only the cluster heads communicate with the fusion centre, reducing the communication cost and also the risk of congestion.

Behnke et al. [13] address issues arising with the clustered version of the distributed least squares algorithm presented in [4]. They report that the algorithm does not scale well with an increasing number of nodes and on large networks does not work at all due to the assumption that each node can communicate with all cluster heads which distribute the precomputed parts of the solution. They develop the scalable distributed least squares (sDLS) algorithm to overcome these drawbacks by limiting the communication of each node to its cluster head. To achieve this, each node is provided with individual precomputed data, in turn reducing the size of the data transferred to each node and also the computations to be performed by each node. Communication and computation costs are therefore independent of the network size and enable scalability of the algorithm also in large networks.

Shakibian and Charkari [14] propose a clustered, multi-swarm version of the particle swarm optimisation algorithm (MMS-PSO) for solving a least squares problem as a minimisation problem. Each cluster head manages the member nodes acting as a sub-swarm of the process. They also use a fusion centre to get the final global result from all cluster heads through weighted averaging. The authors claim that their method decreases the latency through clustering and converges faster than a fusion centre approach.

Summarising, clustering reduces but does not eliminate the risk of a single point of failure affecting the entire network. The cluster heads usually have to be more powerful than the other nodes to be able to handle the higher volume of messages received. If a cluster head fails, the complete area covered by the cluster and its data are lost until a new cluster head takes over.

## 2.3 Truly Distributed Approaches

The most decentralised approach is to limit the communication of the nodes to their immediate neighbourhood (defined by the communication range). Each communication partner has to be reachable in a single hop as multi-hop communication would incur additional overhead through routing and thus increase the energy consumption of the resource restricted nodes.

Zhou et al. [15] propose a distributed least squares solver which they claim is robust against reported node failures. The algorithm is designed for $m = 1$ and higher dimensions are not considered in [15]. The distributed iterative algorithm exchanges the values of $A$ and $b$ with the neighbours and updates them using a Metropolis weight based on the degree of the node's neighbours, which are determined before the iterative algorithm initialises. In the event of a node failure, convergence is still guaranteed, but the result will no longer be correct. Therefore, the authors extend their algorithm, trying to reduce the magnitude of the occurring error. A disadvantage is that node failures have to be detectable. Once detected, the weights used in the computation have to be updated throughout the network, which poses a global updating problem requiring communication across the entire network. In the event of a node failure, the magnitude of the error depends on the network topology. Although the algorithm presented in [15] is truly distributed, we do not consider it in our analysis and in our simulations because it is restricted to the special case $m = 1$.

Sayed et al. [2,16,17] propose a diffusion-based least mean square estimator (diffLMS) using steepest-descent iterations for solving the normal equations. Diffusion strategies are seen as an alternative to consensus strategies for distributed optimisation problems, both limiting the communication to the neighbourhood. $A$ and $b$ are both distributed row-wise. In each iteration, diffLMS consists of two main steps, an adaption step and a combination step, and delivers an estimate of the solution $x$ in each node. The authors provide two variants of their algorithm, adapt-then-combine (ATC) and combine-then-adapt (CTA), which differ in the order of these computation steps (for details, see Section 4).

Another fully distributed approach is the distributed least mean squares method (D-LMS) by Schizas, Mateos and Giannakis [18,19,20]. D-LMS is based on Lagrange multipliers and uses the least squares residual and the difference between the estimates of $x$ from the neighbourhood in a correction step to compute the least squares solution iteratively. The data distribution of $A$ and $b$ is again row-wise. At each step an estimate for the solution $x$ is available in each node. D-LMS communicates twice in each iteration, once to broadcast the current estimate to all neighbours and a second time to send individual correction vectors to each neighbour (single-hop unicast – for details, see Section 4).

## 3 A Push-Sum-Based Least Squares Solver

In this section, we introduce the Push-Sum Distributed Least Squares Solver (PSDLS), shown in Algorithm 1, for problem (1). The matrix $A$ and the vector $b$ are distributed row-wise across the participating nodes. The parts of $A$ and $b$ available locally at node $u$ will therefore be denoted by $A^u$ and $b^u$, respectively. The solution $x$ is approximated at each node. The local instance of a vector $v$ which occurs at every node $u$ will be referred to as $v_u$, and $v_u(i)$ refers to the $i^{\text{th}}$ element of $v_u$. In particular, $x_u$ refers to the approximation of the entire solution vector $x$ at node $u$. The algorithm does not require any knowledge about the global topology of the network and it does not assume any specific connections between the nodes. Each node only needs to know its neighbours. In such a setup, the push-sum algorithm [5] provides a truly distributed way for summing or averaging values across the nodes of the network. If each node knows the total number of nodes $N$ in the network, then the sum of the values over all nodes can be computed using distributed averaging. Note that $N$ can also be estimated in a truly distributed way [21]. Alternatively, the push-sum algorithm can be used to compute the sums directly without the need to know $N$ at every node. However, based on our experience, this variant leads to slightly slower convergence.

**Algorithm 1.** Push-Sum Distributed Least Squares Solver (PSDLS)

```
Input: $A \in \mathbb{R}^{n \times m}$ with $n > m$, $b \in \mathbb{R}^n$, both distributed row-wise over $N$ nodes
Output: $x_u \in \mathbb{R}^m$ on every node
1: in each node $u$ do
2:     $[Q^u, R_u] \leftarrow$ vdmGS($A^u$)
3:     $z_u \leftarrow$ dmmv($Q^{u\top}, b^u$)
4:     $x_u \leftarrow$ solve $R_u x_u = z_u$                    ▷ local
```

PSDLS is a direct least squares solver first computing a distributed QR factorisation of $A$ (line 1.2[1]) and subsequently solving *locally* a linear system with the triangular matrix $R_u$ at every node (line 1.4). For the distributed QR factorisation we use the gossip-based distributed modified Gram-Schmidt orthogonalisation method *vdmGS* introduced in [22,23]. vdmGS returns the orthonormal matrix $Q \in \mathbb{R}^{n \times m}$ distributed row-wise (denoted by $Q^u$) and the complete upper-triangular matrix $R \in \mathbb{R}^{m \times m}$ in every node (denoted by $R_u$). Consequently, $Q^\top$ is distributed column-wise across the nodes. To compute the right-hand side of the linear system (line 1.3), the distributed matrix-vector multiplication *dmmv* described in [23] is used, which accepts the matrix argument

[1] Line x.y refers to line y in Algorithm x.

distributed column-wise and the vector argument distributed row-wise. The solution of the linear system (back substitution) can be computed locally and does not need any further communication with the other nodes because every node has its local estimate of $R$. At the end of the algorithm, each node $u$ has its own local approximation $x_u$ of the solution of the least squares problem (1).

## 4 Communication Cost of Distributed LS Solvers

We compare the communication cost of the novel PSDLS method, both variants of diffLMS described in [16] and D-LMS described in [20] in terms of number of messages and amount of data sent per node.

*diffLMS.* There are different versions of the diffLMS algorithm aside from the order of execution in ATC and CTA mentioned previously. diffLMS can also exchange the observations $b^u$ and matrix rows $A^u$ with the neighbouring nodes to improve the estimate of the solution. This requires an additional step for exchanging the information which increases the communication cost. For better comparison with [16], we will limit the analysis to the versions without the additional information exchange.

In the ATC version of the diffLMS method, shown in Algorithm 2, each node $u$ first computes an intermediate value $\psi_u \in \mathbb{R}^m$, which adds a step-size $\mu$ of the least squares residual to the current estimation of $x_u$, where $A^u$ and $b^u$ correspond to the rows of $A$ and $b$ available locally on node $u$. The intermediate value $\psi_u$ is subsequently broadcast to the local neighbourhood $D_u$. Each node then updates its estimate of $x_u$ with a weighted sum of all received $\psi_i$ ($i \in D_u$), and its own $\psi_u$, the weights being denoted as $\omega_u(i)$. A proof of convergence and several possible weighting matrices are given in [16].

The CTA variant of diffLMS performs exactly the same operations but in a different order. The intermediate values $\psi_u$ are first broadcast to the neighbourhood, then each node computes its estimate of $x_u$ and in the last step the new intermediate value $\psi_u$. According to [2, p.31], *"... the difference between the*

**Algorithm 2.** Diffusion Least Mean Square (diffLMS) - ATC and CTA

**Input:** $A \in \mathbb{R}^{n\times m}$ with $n > m, b \in \mathbb{R}^n$, both distributed row-wise over $N$ nodes
For all nodes $u$: $x_u$ and $\psi_u$ initialised with zero
**Output:** $x_u \in \mathbb{R}^m$ on every node

**Adapt-then-Combine (ATC)**
```
1: in each node $u$ do
2:     while not converged do
3:         $\psi_u \leftarrow x_u + \mu A^{u\top}(b^u - A^u x_u)$
4:         Broadcast $\psi_u$ to $D_u$
5:         $x_u \leftarrow \omega_u(u)\psi_u + \Sigma_{i\in D_u}\omega_u(i)\psi_i$
6:     end while
```

**Combine-then-Adapt (CTA)**
```
1: in each node $u$ do
2:     while not converged do
3:         Broadcast $\psi_u$ to $D_u$
4:         $\psi_u \leftarrow \omega_u(u)x_u + \Sigma_{i\in D_u}\omega_u(i)x_i$
5:         $x_u \leftarrow \psi_u + \mu A^{u\top}(b^u - A^u \psi_u)$
6:     end while
```

**Algorithm 3.** Distributed Least-Mean Squares Solver (D-LMS)

```
Input: $A \in \mathbb{R}^{n \times m}$ with $n > m, b \in \mathbb{R}^n$, both distributed row-wise over $N$ nodes
  For all $u$ and $\forall i \in D_u$: $x_u$ and $v_u^i$ initialised with zero
Output: $x_u \in \mathbb{R}^m$ on every node
1: in each node $u$ do
2:     while not converged do
3:         Broadcast $x_u$ to $D_u$
4:         for each node $i \in D_u$ do
5:             $v_u^i = v_u^i + \frac{c}{2}(x_u - x_i)$
6:         Send $v_u^i$ to each corresponding node $i \in D_u$
7:         $x_u = x_u + \mu[2A^{u\top}(b^u - A^u x_u) - \Sigma_{i \in N_u}(v_u^i - v_i^u) - c\Sigma_{i \in N_u}(x_u - x_i)]$
8:     end while
```

*implementations lies in which variable we choose to correspond to the updated weight estimate.".* In ATC, $x_u$ is the result of the combination step (line 2.5 of ATC), in CTA it is the result of the adaption step (line 2.5 of CTA). However, mathematically and numerically this does not result in the same solution.

*D-LMS.* The D-LMS method is shown in Algorithm 3. A node $u$ first broadcasts its current estimate $x_u$ to its neighbourhood $D_u$ (line 3.3). Then an individual correction vector $v_u^i$ is computed for each neighbour $i \in D_u$ using the received estimation $x_i$ and its own estimation $x_u$ (line 3.5). These values are then sent to each corresponding node $i$. In the last step of each iteration (line 3.7), the new estimate $x_u$ is computed using a least squares residual from $A^u$ and $b^u$, the locally available parts of $A$ and $b$, and the correction terms $v_u^i$ and $v_i^u$ received from the neighbourhood. This term is added to the current $x_u$ and weighted with a step-size parameter $\mu$ resulting in an estimate $x_u$ of the solution $x$ in each node. Proof of convergence is given in [18].

## Comparison of Communication Cost

The cost of a broadcast to all neighbours ("local broadcast") depends on the topology and on the type of connection. Therefore, we introduce the broadcasting parameter $B(d)$ for denoting the number of messages required for broadcasting to $d$ neighbours. In a wireless setting, a single message is required to perform a broadcast to all neighbours, thus $B(d) = 1$. However, in a setting with point-to-point communication (e. g., wired connections), $d$ messages are required for sending a message to $d$ neighbours, thus $B(d) = d$. For a global broadcast beyond the neighbourhood in any network other than a fully connected one, additional messages are needed for multi-hop message relaying over intermediate nodes.

The communication patterns and costs for ATC and CTA are identical. In each iteration, each node $u$ broadcasts a vector of size $m$ to its neighbourhood $D_u$. In $k_1$ iterations, node $u$ sends $k_1 B(|D_u|)$ messages. D-LMS requires communication in two of its steps. In line 3.3, a local broadcast is required to distribute the vector $x_u$ of size $m$ to the neighbours. Line 3.6 sends $|D_u|$ individual messages of size

**Table 1.** Comparison of the communication cost for diffLMS, D-LMS and PSDLS

| **Algorithm** | **Number of messages sent per node** | **Total amount of data sent per node** |
|---|---|---|
| diffLMS | $k_1\, B(\lvert D_u\rvert)$ | $k_1\, B(\lvert D_u\rvert)\, m$ |
| D-LMS | $k_2\,(\,B(\lvert D_u\rvert) + \lvert D_u\rvert\,)$ | $k_2\,(\,B(\lvert D_u\rvert) + \lvert D_u\rvert\,)\, m$ |
| PSDLS | $2mR$ | $\frac{1}{2}\left(m^2 + 7m\right) R$ |

$m$ to distribute the correction term. This results in $k_2(B(|D_u|) + |D_u|)$ messages and $k_2(B(|D_u|) + |D_u|)m$ data values sent per node.

Although PSDLS is not an iterative method, we have to consider the number of rounds $R$ required by each push-sum algorithm. Note that in practice $R$ may vary slightly for different push-sum calls due to the randomisation. In the distributed QR decomposition, for the first $m-1$ columns of the matrix $A$ two push-sum calls have to be executed, the first one summing scalars and the second one summing vectors. In column $l$ of $A$ the length of these vectors is $m-l$. For column $m$ only one scalar push-sum call has to be executed. The matrix-vector product $Q^\top b$ requires one more push-sum call on vectors of length $m$. Consequently, the number of messages sent per node is $2mR$. In each push-sum call, the values *and* a weight have to be transmitted [5].

Table 1 summarises the analytical results of this section. We conclude that independently of the number of iterations $k_1$ and $k_2$, D-LMS sends $|D_u|$ more messages and more data per iteration than diffLMS. For comparing the communication cost, information about the number of iterations $k_1$ and $k_2$ required by diffLMS and D-LMS, respectively, and the number of push-sum rounds $R$ required by PSDLS is necessary. As our simulation results in Section 5 illustrate, these quantities differ significantly across the three methods.

## 5 Experiments

The simulation results presented in this section demonstrate the different convergence speeds in terms of average number of messages sent per node and therefore provide some qualitative insight into typical values of $k_1$, $k_2$ and $R$ for the algorithms compared in this paper. Our simulations are based on Matlab implementations of the algorithms. The implementation of the push-sum algorithm is round-based and synchronised. The neighbours are selected at random from a uniform distribution. For all methods, $A$ and $b$ are distributed row-wise over all $N$ nodes. Without loss of generality, we consider the special case $n = N$, i. e., each node holds one row of $A$ and one element of $b$. Like in [16], the relative degree weight matrix was used for both diffLMS and D-LMS.

In order to evaluate the accuracy of the approximate solution $x_u$ computed by the algorithms, we evaluated the relative error

$$\max_{u=1,..,N} \|x_u - x^*\|_\infty / \|x^*\|_\infty, \tag{2}$$

where $x^*$ is the solution computed sequentially in Matlab.

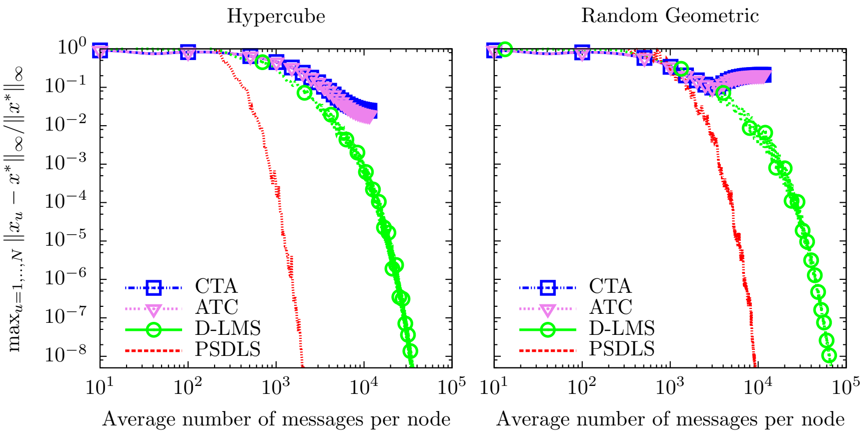

**Fig. 1.** Comparison for $N = n = 64, m = 8$ on different topologies: hypercube (left) and random geometric (right). The step-sizes are $\mu_{\text{ATC}} = \mu_{\text{CTA}} = 0.01$ and $\mu_{\text{D-LMS}} = 0.2$.

diffLMS and D-LMS are both iterative methods, whereas the PSDLS is a direct method with an iterative building block (the push-sum algorithm) in each step. For a fair comparison of the methods, the instances of the push-sum algorithm in PSDLS were not terminated based on reaching a predefined accuracy, but based on a predefined maximum number of rounds.

The behaviour of diffLMS and D-LMS strongly depends on the choice of the step-size parameter $\mu$. Based on our experience, in particular the convergence speed of diffLMS is very sensitive to the choice of $\mu$, and for bad choices of $\mu$ the methods even diverge. The best choice for $\mu$ in terms of convergence speed seems to vary greatly with $m$, the topology and the average node degree. Unfortunately, the literature does not give any guidance on how to choose $\mu$. Thus, we performed extensive simulations across a wide range of values for $\mu$ and chose the values at which the respective algorithm eventually achieves the highest accuracy.

Figure 1 shows the convergence behaviour of the different algorithms for $N = 64$ nodes arranged in a hypercube and as a random geometric graph on the unit square with a communication radius 0.2. The horizontal axis shows the average number of messages sent per node and the relative error (2) achieved for this number of messages sent per node is plotted on the vertical axis. The experiments show that the diffLMS methods do not reach the targeted accuracy of $10^{-8}$ and after 12000 messages only achieve an accuracy of $10^{-2}$ on a hypercube. On a random geometric graph diffLMS diverges at around 3100 messages and does not even reach $10^{-1}$. On a hypercube network, the D-LMS algorithm achieves an accuracy of $10^{-8}$, but requires around 32600 messages to be sent per node. The PSDLS method converges significantly faster than the other algorithms requiring only about 1950 messages per node to reach an accuracy of $10^{-8}$, which is a factor of 16 less than D-LMS. The amount of data sent per node is also significantly lower for PSDLS, sending only 5400 values compared to 261000 values sent by

D-LMS. Similar behaviour can be observed for the random geometric graph. PSDLS converges more than 7 times faster than D-LMS and sends only 0.05% of the data sent by D-LMS.

## 6 Conclusion

We surveyed existing distributed least squares solvers and classified them based on their communication pattern. We introduced a novel truly distributed least squares solver PSDLS based on the push-sum algorithm, which limits the communication to the immediate neighbourhood of each node and does not require a fusion centre or clustering.

We analysed and compared the communication cost of all existing truly distributed methods in terms of the number of messages and the amount of data sent per node. Numerical simulations showed that the number of messages per node required for a solution accuracy of $10^{-8}$ is more than a factor of seven lower for the novel PSDLS algorithm than for the other truly distributed methods.

Future work will consider fault tolerance in distributed least squares solvers.

**Acknowledgement.** This work was partly supported by the Austrian Science Fund (FWF): S 10608-N13 (NFN SISE).

## References

1. Reise, G., Matz, G., Gröchenig, K.: Distributed field reconstruction in wireless sensor networks based on hybrid shift-invariant spaces. IEEE Transactions on Signal Processing 60(10), 5426–5439 (2012)
2. Sayed, A.H.: Diffusion adaptation over networks. In: Academic Press Library in Signal Processing, vol. 3, pp. 323–454. Academic Press, Elsevier (2014)
3. Shi, L., Song, W.Z., Xu, M., Xiao, Q., Kamath, G., Lees, J., Xing, G.: Imaging seismic tomography in sensor network. In: IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 304–306 (2013)
4. Reichenbach, F., Born, A., Timmermann, D., Bill, R.: A distributed linear least squares method for precise localization with low complexity in wireless sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 514–528. Springer, Heidelberg (2006)
5. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 482–491 (2003)
6. Gansterer, W.N., Niederbrucker, G., Straková, H., Schulze Grotthoff, S.: Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation. Journal of Computational Science 4(6), 480–488 (2013)
7. Niederbrucker, G., Straková, H., Gansterer, W.N.: Improving fault tolerance and accuracy of a distributed reduction algorithm. In: SC Companion: High Performance Computing, Networking, Storage and Analysis, pp. 643–651 (2012)
8. Khan, M.I., Gansterer, W.N., Haring, G.: Static vs. mobile sink: The influence of basic parameters on energy efficiency in wireless sensor networks. Computer Communications 36(9), 965–978 (2013)

9. Le Borgne, Y.A., Nowe, A., Abughalieh, N., Steenhaut, K.: Distributed regression for high-level feature extraction in wireless sensor networks. In: 2010 Seventh International Conference on Networked Sensing Systems (INSS), pp. 249–252 (2010)
10. Shi, L., Song, W.Z., Kamath, G., Xing, G., Liu, X.: Distributed least-squares iterative methods in networks: A survey. Submitted to Computing Journal (2013)
11. Renaut, R.A.: A parallel multisplitting solution of the least squares problem. Numerical Linear Algebra with Applications 5(1), 11–31 (1998)
12. Yang, L., Brent, R.: Parallel MCGLS and ICGLS methods for least squares problems on distributed memory architectures. The Journal of Supercomputing 29(2), 145–156 (2004)
13. Behnke, R., Salzmann, J., Lieckfeldt, D., Timmermann, D.: SDLS - Distributed least squares localization for large wireless sensor networks. In: International Conference on Ultra Modern Telecommunications & Workshops, pp. 1–6 (2009)
14. Shakibian, H., Charkari, N.: MMS-PSO for distributed regression over sensor networks. In: IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pp. 68–73 (2010)
15. Zhou, Q., Kar, S., Huie, L., Poor, H.V.: Robust distributed least-squares estimation in sensor networks with node failures. In: IEEE Global Telecommunications Conference, pp.1–6 (2011)
16. Cattivelli, F., Sayed, A.: Diffusion LMS strategies for distributed estimation. IEEE Transactions on Signal Processing 58(3), 1035–1048 (2010)
17. Tu, S.Y., Sayed, A.: Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks. IEEE Transactions on Signal Processing 60(12), 6217–6234 (2012)
18. Schizas, I.: Consensus in ad hoc WSNs with noisy links - Part II: Distributed estimation and smoothing of random signals. IEEE Transactions on Signal Processing 56(4), 1650–1666 (2008)
19. Mateos, G., Schizas, I.D., Giannakis, G.B.: Performance analysis of the consensus-based distributed LMS algorithm. EURASIP Journal on Advances in Signal Processing 2009(1), 68:6–68:6 (2009)
20. Schizas, I.D., Mateos, G., Giannakis, G.B.: Distributed LMS for consensus-based in-network adaptive processing. IEEE Transactions on Signal Processing 57(6), 2365–2382 (2009)
21. Sluciak, O., Rupp, M.: Network size estimation using distributed orthogonalization. IEEE Signal Processing Letters 20(4), 347–350 (2013)
22. Straková, H., Gansterer, W.N., Zemen, T.: Distributed QR factorization based on randomized algorithms. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part I. LNCS, vol. 7203, pp. 235–244. Springer, Heidelberg (2012)
23. Straková, H., Gansterer, W.N.: A distributed eigensolver for loosely coupled networks. In: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 51–57 (2013)