Scheduling in data gathering networks with background communications

Joanna Berlińska¹

Published online: 20 February 2020
© The Author(s) 2020

Abstract

In this work, we study scheduling in star data gathering networks with background communications. The worker nodes of the network hold datasets that have to be transferred directly to the base station. The communication speed of each link changes with time, due to other applications using the network, independently of the speeds of other links. Our goal is to gather all data within the minimum time. We show that while the preemptive version of this problem can be solved in polynomial time, the non-preemptive variant is strongly NP-hard. We propose for it an exact exponential algorithm based on dynamic programming, several polynomial time heuristics and local search algorithms. Theoretical performance guarantees and the existence of dominance relations between the heuristics are studied. The performance of the proposed algorithms is also tested in a series of computational experiments.

Keywords Scheduling · Data gathering networks · Variable communication speed

1 Introduction

Scheduling for data gathering is becoming increasingly important due to a growing number of applications producing data in distributed locations. Data gathering wireless sensor networks are used in environmental, military, health and home applications (Akyildiz et al. 2002). Complex computations are executed in clouds or grids, and hence, the obtained results are scattered on large numbers of processing nodes. These data often have to be gathered on a single machine for merging and further processing. The efficiency of the data gathering process influences the performance of the whole distributed application.

Scheduling in data gathering networks was studied on the grounds of divisible load theory by Choi and Robertazzi (2008) and Moges and Robertazzi (2006). The total amount of measurement data to be collected was known, but it was possible to divide the measurements arbitrarily between dif-

ferent network nodes. Thus, the problem was what quantities of work to assign to individual workers, and how to organize sending the data to the base station, in order to minimize the total time of gathering the data. Later research concentrated on designing scheduling algorithms for gathering datasets of fixed sizes from the network nodes. Berlińska (2014) analyzed maximizing the lifetime of a network with limited memory of nodes acquiring the data. Algorithms minimizing data gathering time were proposed for networks with data compression (Berlińska 2015; Luo et al. 2018a, b) and networks with limited base station memory (Berlińska 2020). Minimizing the maximum dataset lateness was studied by Berlińska (2018a). All the above works share a common assumption that the network performance is constant. However, in reality the communication parameters of a network may change with time because of sharing the links with other applications, maintenance activities, etc. Scheduling in such networks with background communications is the subject of this paper.

We assume that the communication rate of each link may change at any time, independently of the speeds of other links. We study the simple scenario of a network that is shared by two applications and implements QoS percentage-based policing (Szigeti et al. 2013). Thus, the speed of a link perceived by the analyzed application drops by a constant factor when it is shared. We prove that even under

This research has been partially supported by the National Science Centre, Poland, Grant 2016/23/D/ST6/00410.

✉ Joanna Berlińska
Joanna.Berlinska@amu.edu.pl

¹ Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Uniwersytetu Poznańskiego 4, 61-614 Poznań, Poland

Fig. 1 Optimum schedules for the instance in Example 1. Bold segments on the time axes indicate intervals in which given links are loaded. The numbers in boxes representing communications are the sizes of sent data. **a** Preemptive scheduling and **b** non-preemptive scheduling

this assumption, the problem of minimizing data gathering time is strongly NP-hard if no preemptions are allowed. We propose an exact exponential algorithm, a group of simple heuristics and local search algorithms. Their performance is analyzed both theoretically and experimentally. Our preliminary results concerning the analyzed problem can be found in Berlińska (2018b).

The rest of this paper is organized as follows. In Sect. 2 we formulate the scheduling problem. In Sect. 3 we show that if preemptions are allowed, then it can be solved in polynomial time. The non-preemptive version of our problem is proved to be strongly NP-hard in Sect. 4. A dynamic programming algorithm for solving it is presented in Sect. 5, and heuristics are proposed in Sect. 6. The results of computational experiments on the algorithms performance are presented in Sect. 7. The last section is dedicated to conclusions.

2 Problem formulation

We study a star data gathering network which consists of m worker nodes $P_1, \dots, P_m$ and a single base station P_0 . Each worker P_i holds dataset D_i of size α_i , which has to be transferred directly to the base station. At most one node can communicate with the base station at a time. The communication rate of node P_i depends on the corresponding link being used by other applications. We will be calling a link *loaded* if it is used by background communications at a given time, and *free* in the opposite case. Transferring one unit of data from P_i to P_0 over a free link takes time c_i , for $i = 1, \dots, m$. A loaded link becomes δ times slower, for some fixed rational $\delta > 1$. Thus, sending a unit of data over a loaded link between P_i and P_0 takes time δc_i .

The maximum time that may be necessary to gather data from all worker nodes is $\bar{T} = \delta \sum_{i=1}^m c_i \alpha_i$. The communication speed changes are described in the following way. For each node P_i , we are given a set of n_i disjoint time intervals $[t'_{i,j}, t''_{i,j}]$ (where $j = 1, \dots, n_i$, $t'_{i,j} < t'_{i,j+1}$ for $j < n_i$, and $t'_{i,n_i} < \bar{T}$), in which the corresponding communica-

tion link is loaded. The total number of such intervals is $n_1 + \dots + n_m = n$.

The scheduling problem is to organize the communication in the network so that all datasets are transferred in the shortest possible time T . Our main interest lies in non-preemptive scheduling, i.e., we assume that each dataset has to be transferred in a single message. In Sect. 3 we briefly analyze the case when this assumption is removed.

Note that sending data of size α at unit communication time c is equivalent to sending data of size $c\alpha$ at unit communication time 1. Therefore, from now on we assume without loss of generality that $c_i = 1$ for $i = 1, \dots, m$.

To finish this section, let us analyze the following numerical example.

Example 1 Let $m = 3$, $\delta = 2$, and let the datasets sizes be $\alpha_1 = 3$, $\alpha_2 = 1$, $\alpha_3 = 2$. Suppose that the first link is loaded in time intervals $[1, 2)$, $[3, 4)$ and $[5, 8)$, the second link is loaded in intervals $[0, 2)$ and $[3, 4)$, and the third link is loaded in intervals $[0, 2)$ and $[4, 6)$ (see Fig. 1).

The length of the optimum preemptive schedule, presented in Fig. 1a, is $6\frac{1}{2}$. Indeed, all links are loaded in interval $[1, 2)$, and hence, it is not possible to send all data in a shorter time.

In the case when preemptions are not allowed, the minimum schedule length is $7\frac{1}{2}$. The optimum order of datasets is (D_1, D_2, D_3) , as shown in Fig. 1b. It is easy to check that sending datasets in a different order results in a longer schedule. $\square$

3 Preemptive scheduling

In this section, we show that if dataset transfers can be preempted, then our problem can be solved in polynomial time, using the method proposed by Berlińska (2016). To make this paper self-contained, we describe the algorithm in full in the proofs of Lemma 1 and Proposition 1.

Let $\tau_1 < \tau_2 < \dots < \tau_N$ be all the moments when the communication rate of at least one worker changes. Additionally, let $\tau_0 = 0$ and $\tau_{N+1} = \infty$. Note that $N = O(n)$. The communication speed of node P_i in interval $[\tau_j, \tau_{j+1})$

Fig. 2 Schedule structure for the proof of Proposition 2. Bold segments on the time axis indicate intervals in which links $1, \dots, 3s$ are loaded. $D_i \sim S_j$ denotes the set of datasets corresponding to S_j , i.e., $\{D_i : i \in S_j\}$

is constant and equal to $1/c_{i,j}$, where $c_{i,j} = 1$ if the corresponding link is free, and $c_{i,j} = \delta$ if it is loaded in this interval.

Lemma 1 *If it is known that the optimum schedule length T belongs to interval $[\tau_k, \tau_{k+1})$ for a given $k \in \{0, \dots, N\}$, then our preemptive scheduling problem can be solved in polynomial time.*

Proof Let $x_{i,j}$ be rational variables representing the amounts of data sent by node P_i in interval $[\tau_j, \tau_{j+1})$, for $1 \leq i \leq m$, $0 \leq j \leq k$. Our problem can be formulated as the following linear program.

$$\text{minimize } T \quad (1)$$

$$\sum_{j=0}^k x_{i,j} = \alpha_i \quad \text{for } i = 1, \dots, m \quad (2)$$

$$\sum_{i=1}^m c_{i,j} x_{i,j} \leq \tau_{j+1} - \tau_j \quad \text{for } j = 0, \dots, k-1 \quad (3)$$

$$\sum_{i=1}^m c_{i,k} x_{i,k} \leq T - \tau_k \quad (4)$$

In the above program we minimize the data gathering completion time T . By constraints (2) each node transfers all its data to the base station. Inequalities (3) and (4) guarantee that the communications fit in the time intervals where they are assigned. Linear program (1)–(4) has $m(k+1)+1 = O(mn)$ variables and $m+k+1 = O(m+n)$ constraints, and hence, it can be solved in polynomial time. The optimum communication schedule can be easily obtained from the values of variables $x_{i,j}$. Namely, in each interval $[\tau_j, \tau_{j+1})$ we schedule consecutively communications from nodes $P_1, \dots, P_m$ of sizes $x_{1,j}, \dots, x_{m,j}$ correspondingly, starting at time τ_j . Thus, the optimum schedule can be constructed in polynomial time. $\square$

Proposition 1 *The preemptive version of our problem can be solved in polynomial time.*

Proof The algorithm consists in using binary search to find the smallest k for which our problem with the additional assumption that $T \in [\tau_k, \tau_{k+1})$ has a solution. The number of binary search iterations is $O(\log n)$, and each iteration requires polynomial time by Lemma 1. Thus, the whole algorithm is also polynomial. $\square$

4 Computational complexity

From now on, we will analyze only the non-preemptive version of our scheduling problem. In this section we show that it is computationally hard.

Proposition 2 *Non-preemptive scheduling to minimize data gathering time in networks with background communications is strongly NP-hard.*

Proof We will use a pseudopolynomial transformation from the following strongly NP-complete 3-partition problem (Garey and Johnson 1979). Given a positive integer K and a set of $3s$ elements $\{1, \dots, 3s\}$, each i of which has a positive integer size x_i , such that $K/4 < x_i < K/2$ for $i \in S = \{1, 2, \dots, 3s\}$, and $\sum_{i=1}^{3s} x_i = sK$, is it possible to partition the index set S into s disjoint sets $S_1, S_2, \dots, S_s$ such that for each $1 \leq j \leq s$, $\sum_{i \in S_j} x_i = K$?

Given an arbitrary instance of 3-partition, we construct the following instance of our scheduling problem. The network consists of $m = 4s - 1$ nodes. The dataset sizes are $\alpha_i = x_i$ for $i = 1, \dots, 3s$, and $\alpha_i = 1$ for $i = 3s + 1, \dots, 4s - 1$. Parameter δ can be an arbitrary number greater than 1. The links used by nodes $P_1, \dots, P_{3s}$ are loaded in intervals $[jK + j - 1, j(K + 1))$, for $j = 1, \dots, s - 1$. The links used by nodes $P_{3s+1}, \dots, P_{4s-1}$ are always free.

We will show that a schedule of length at most $T = \sum_{i=1}^m \alpha_i = sK + s - 1$ exists if and only if the corresponding instance of 3-partition is a “yes” instance.

Indeed, suppose that such a schedule exists. Note that since $\delta > 1$, all datasets have to be sent over free links. Thus, datasets $D_1, \dots, D_{3s}$ have to be transferred in s intervals $[(j-1)(K+1), jK+j-1)$ of length K each, for $j = 1, \dots, s$ (see Fig. 2). As $K/4 < \alpha_i < K/2$ for $i = 1, \dots, 3s$ and $\sum_{j=1}^{3s} \alpha_j = sK$, each of these intervals has to contain transfers of exactly 3 datasets of total size K . Thus, by defining subset S_j as the set of indices i such that datasets D_i are sent in interval $[(j-1)(K+1), jK+j-1)$, for $j = 1, \dots, s$, we obtain the required 3-partition.

Conversely, if the 3-partition exists, we construct the following schedule (cf. Fig. 2). Datasets D_i such that $i \in S_j$ are transferred one by one in interval $[(j-1)(K+1), jK+j-1)$. For $i = 3s + 1, \dots, 4s - 1$, dataset D_i is transferred in interval $[(i-3s-1)(K+1)+K, (i-3s)(K+1))$. The schedule length is $sK + s - 1 = T$.

Thus, a schedule of length at most T exists if and only if the corresponding 3-partition exists. It is clear that the above transformation can be done in polynomial time. Moreover, the size of the analyzed 3-partition instance is polynomial with respect to the obtained instance of our scheduling problem. Finally, the largest numerical value occurring in the instance of our problem is pseudopolynomial with respect to the 3-partition instance. Thus, our transformation is not only polynomial, but also pseudopolynomial, and hence, the analyzed scheduling problem is strongly NP-hard. $\square$

5 Dynamic programming algorithm

In this and the following section, we design scheduling algorithms for solving the analyzed problem. Let us note that any schedule containing idle times will be shortened if we keep its dataset order, but remove unnecessary delays. Therefore, we will only analyze algorithms that construct schedules without idle times, which can be represented by permutations of consecutively sent datasets.

As our problem is computationally hard, we propose the following exponential algorithm **DP** for solving it to the optimum. Let $\tau(D_i, t)$ be the time necessary to transfer dataset D_i , starting at moment t . Note that $\tau(D_i, t)$ can be computed in $O(n_i)$ time by scanning the list of loaded intervals for node P_i . For each subset $\mathcal{D} \subset \{D_1, \dots, D_m\}$, we can compute the shortest time $T(\mathcal{D})$ in which the datasets from $\mathcal{D}$ can be transferred to the base station, using the following recursive formula:

$$T(\mathcal{D}) = \begin{cases} 0 & \text{for } \mathcal{D} = \emptyset, \\ \min_{D_i \in \mathcal{D}} \{T(\mathcal{D} \setminus \{D_i\}) + \tau(D_i, T(\mathcal{D} \setminus \{D_i\}))\} & \text{for } \mathcal{D} \neq \emptyset. \end{cases}$$

We use the dynamic programming approach to compute $T(\mathcal{D})$ for all subsets $\mathcal{D}$. Naturally, the subsets have to be processed in the order of nondecreasing numbers of elements. To this end, we use a first-in-first-out queue. At the beginning, the queue holds only the empty set. In each step i of the algorithm, we pop a subset $\mathcal{E}_i$ from the front of the queue and update values $T(\mathcal{E}_{ij})$ for all subsets $\mathcal{E}_{ij} = \{\mathcal{E}_i \cup \{D_j\}\}$ such that $D_j \notin \mathcal{E}_i$. If $T(\mathcal{E}_{ij})$ is computed for the first time, we append subset $\mathcal{E}_{ij}$ at the end of the queue. The algorithm completes when the queue becomes empty.

The minimum schedule length is $T(\{D_1, \dots, D_m\})$, and the optimum dataset sequence can be easily tracked. The running time of this algorithm is $O((m+n)2^m)$.

We will now analyze the possibility of improving the running time of DP by reducing the analyzed state space using optimality properties.

Proposition 3 *Suppose there exist a set of $k > 1$ nodes $P_{i_1}, \dots, P_{i_k}$ such that $i_1 < \dots < i_k$, $\alpha_{i_1} = \dots = \alpha_{i_k}$,*

and all these nodes are loaded in the same time intervals. Then, subsets $\mathcal{D}$ such that $D_{i_q} \in \mathcal{D}$ but $D_{i_p} \notin \mathcal{D}$ for some $1 \leq p < q \leq k$, can be removed from the state space analyzed by the DP algorithm.

Proof Nodes $P_{i_1}, \dots, P_{i_k}$ are identical from the point of view of our scheduling problem, and hence, it can be assumed without loss of generality that they communicate in the order of increasing indices i_j . $\square$

Proposition 3 shows that the state space of DP can be reduced for instances that contain sets of nodes which hold datasets of the same sizes and follow the same pattern of loaded intervals. Unfortunately, this assumption is very strong, and hence, for most instances, the running time of DP cannot be improved using this property.

We will now show that if two nodes have the same dataset sizes, but different loaded intervals, or if they have identical loaded intervals, but hold datasets of different sizes, then the order in which they should send their messages depends on the remaining nodes.

Proposition 4 *Suppose nodes P_i and P_j are loaded in the same time intervals, but $\alpha_i \neq \alpha_j$. Then, the order in which P_i and P_j communicate with the base station in the optimum schedule depends on the parameters of the remaining nodes.*

Proof Let $m = 3$, $\alpha_1 = 1$, $\alpha_2 = 2$, $\alpha_3 = 1$, and let the first two links be always free. Suppose that the third link is free only in interval $[1, 2)$. Then, the only optimum sequence of messages is (D_1, D_3, D_2) , as it allows to send all data over free links, in time $T = 4$. Contrarily, assume that the third link is free only in interval $[2, 3)$. In this case, the optimum schedule of length 4 can be obtained only by choosing dataset permutation (D_2, D_3, D_1) . Thus, the order in which D_1 and D_2 should be sent, depends on the parameters of node P_3 . $\square$

Proposition 5 *Suppose that $\alpha_i = \alpha_j$, but the lists of loaded intervals of nodes P_i and P_j are not identical. Then, the order in which P_i and P_j communicate with the base station in the optimum schedule depends on the parameters of the remaining nodes, even if the link of P_i is always free, and the link of P_j is always loaded.*

Proof Let $m = 3$, $\alpha_1 = \alpha_2 = \alpha_3 = 1$. Assume that the first link is always free, and the second link is always loaded. If the third link is free only in interval $[1, 2)$, then the only optimum dataset sequence is (D_1, D_3, D_2) , resulting in schedule of length $2 + \delta$. However, if the third link is free only in interval $[\delta, \delta + 1)$, then the optimum schedule of length $2 + \delta$ can be achieved only by dataset permutation (D_2, D_3, D_1) . Hence, the order in which D_1 and D_2 should be transferred, depends on the parameters of node P_3 . $\square$

Note that instances constructed similarly to the ones presented in the proofs of Propositions 4 and 5 can be used to

show that the optimum order of communications of nodes P_i and P_j also depends on the remaining nodes in other cases, e.g., when the link of P_i is always free, the link of P_j is always loaded, and $\alpha_i < \alpha_j$ (or $\alpha_i > \alpha_j$). These results suggest that reducing the state space of DP algorithm using dominance properties may not be possible for the general version of our problem.

6 Heuristic algorithms

Since algorithm DP cannot be used for large instances due to its high computational complexity, we design the following simple heuristics running in $O(m(m+n))$ time, which build dataset sequences step by step, using greedy methods.

1. Algorithm **gTime** always chooses to send the dataset that will be transferred in the shortest time.
2. Algorithm **gRate** selects the dataset that will be sent at the best average communication rate.
3. Algorithm **gSlowtime** chooses the dataset for which the time when data are transferred over a loaded link will be the shortest.

In all these three heuristics ties are broken by selecting a larger dataset, and in the case of equal dataset sizes, by choosing a dataset with a lower number. We also implement algorithm **Rnd**, which constructs a random dataset sequence and computes the corresponding makespan in $O(m+n)$ time.

Finally, we propose a group of local search algorithms with neighborhoods based on dataset swaps. Each of these algorithms starts with a schedule generated by one of the simple heuristics and then applies the following local search procedure. For each pair of datasets, we check whether swapping their positions in the current sequence leads to decreasing the schedule length. The swap that results in the shortest schedule is executed, and the search continues until no further improvement is possible. These algorithms are called **gTimeLocal**, **gRateLocal**, **gSlowtimeLocal** and **RndLocal**. The beginning of each name refers to the heuristic used to construct the initial schedule.

We will now analyze theoretical performance guarantees of heuristics gTime, gRate and gSlowtime. Obviously, local search algorithms deliver solutions that are not worse than the initial schedules obtained by the corresponding simple heuristics.

Proposition 6 *The approximation ratio of any algorithm is at most δ .*

Proof The maximum possible time of transferring the data is $\delta \sum_{i=1}^m \alpha_i$. The optimum schedule is not shorter than $\sum_{i=1}^m \alpha_i$. The claim follows. $\square$

Proposition 7 *Algorithms gTime and gSlowtime have tight approximation ratios equal to δ .*

Proof We will analyze the following instance. Let $m \geq 2$. We have one big dataset of size $\alpha_1 = (m-1)\delta + 2$ and $m-1$ small datasets with $\alpha_2 = \dots = \alpha_m = 1$. The first communication link is free in interval $[0, (m-1)\delta)$ and loaded in interval $[(m-1)\delta, \infty)$. Communication links $2, \dots, m$ follow the opposite pattern: they are loaded in interval $[0, (m-1)\delta)$ and free in interval $[(m-1)\delta, \infty)$.

The maximum time that may be necessary to transfer a small dataset is δ , whereas the time of transferring the big dataset is at least $(m-1)\delta + 2 > \delta$. Hence, algorithm gTime always prefers sending a small dataset to sending the big one.

As the first link is free only in interval of length $(m-1)\delta$, the big dataset must be sent over a loaded link for at least 2δ time units. The maximum possible slowdown time for a small dataset is δ . Thus, algorithm gSlowtime also prefers small datasets.

Therefore, both algorithms gTime and gSlowtime will first send all small datasets in total time $(m-1)\delta$, and then dataset D_1 in time $\delta\alpha_1 = \delta((m-1)\delta + 2)$. This results in schedule length

$$T = (m-1)\delta(\delta+1) + 2\delta. \quad (5)$$

However, in the optimum schedule, dataset D_1 is transferred first, in time $(m-1)\delta + 2\delta$, and then the $m-1$ small datasets are sent in time 1 each. Hence, the minimum schedule length is

$$T^* = (m-1)(\delta+1) + 2\delta. \quad (6)$$

Since

$$\lim_{m \rightarrow \infty} \frac{T}{T^*} = \lim_{m \rightarrow \infty} \frac{(m-1)\delta(\delta+1) + 2\delta}{(m-1)(\delta+1) + 2\delta} = \delta, \quad (7)$$

the approximation ratios of algorithms gTime and gSlowtime are not smaller than δ . By Proposition 6, the claim follows. $\square$

Proposition 8 *Algorithm gRate has a tight approximation ratio of δ .*

Proof Let k be a number such that $k\delta$ is an integer greater than 2, and let $m = k\delta + 1$. Suppose we have to collect one big dataset of size $\alpha_1 = k$ and $k\delta$ small datasets of size $\alpha_2 = \dots = \alpha_{k\delta+1} = 1$. The first link is loaded in interval $[0, (k-1)\delta)$. Links $2, \dots, k\delta+1$ are loaded in intervals $[0, \delta)$ and $[(k-1)\delta + 1, \infty)$. At time 0, each small dataset can be sent at average speed $1/\delta$, and the average speed for sending the big dataset is $k/((k-1)\delta + 1) > 1/\delta$. Thus, algorithm gRate sends dataset D_1 first and finishes this transfer at time

$(k-1)\delta + 1$. Hence, all $k\delta$ small datasets are sent over loaded links in time δ each, and the resulting schedule length is

$$T = (k-1)\delta + 1 + k\delta^2 = k\delta(\delta + 1) - \delta + 1. \quad (8)$$

In the optimum schedule, the small datasets are sent first. The free intervals on the respective links allow to transfer data of size $(k-2)\delta + 1$. Thus, parts of small datasets of total size $2\delta - 1$ have to be sent over loaded links. Consequently, the transfer of all small datasets finishes at time

$$t = (k-2)\delta + 1 + (2\delta - 1)\delta > (k-1)\delta + 1, \quad (9)$$

and the big dataset is sent over a free link, in time k . Hence, the optimum makespan is

$$T^* = t + k = k(\delta + 1) + (2\delta - 1)(\delta - 1). \quad (10)$$

As

$$\lim_{k \rightarrow \infty} \frac{T}{T^*} = \lim_{k \rightarrow \infty} \frac{k\delta(\delta + 1) - \delta + 1}{k(\delta + 1) + (2\delta - 1)(\delta - 1)} = \delta, \quad (11)$$

the approximation ratio of gRate is at least δ . By Proposition 6, we obtain the claim. $\square$

We showed that the tight approximation ratios of algorithms gTime, gRate and gSlowtime are as big as possible. Still, the average performance of these algorithms may be much better. We will evaluate it experimentally in Sect. 7.

It is interesting to compare the algorithms with each other, and determine whether there exist domination relations between them. In the remainder of this section, we show that this is not the case. For each of the three heuristics analyzed here, there exist instances which it solves to the optimum, while the remaining two algorithms do not.

Proposition 9 *There exist instances which are solved to the optimum by gRate, but not by gTime or gSlowtime.*

Proof Let us analyze the instance described in the proof of Proposition 7. We already know that the solutions returned by gTime and gSlowtime are far from the optimum, especially when m is large. At time 0, any small dataset will be sent at average communication speed $1/\delta$, and the large dataset D_1 will be transferred at average speed larger than $1/\delta$. Thus, algorithm gRate starts with sending dataset D_1 , and in consequence, it constructs the optimum schedule. $\square$

Proposition 10 *There exist instances which are solved to the optimum by gTime, but not by gRate or gSlowtime.*

Proof Let $m = 2$, $\alpha_1 = 1$, $\alpha_2 = 2$. Suppose that the first link is always loaded, and the second link is loaded in interval

$[0, \delta)$, and free afterward. The average communication speed of a transfer starting at time 0 is $1/\delta$ if D_1 is sent, and $2/(\delta + 1)$ if D_2 is chosen. Since $2/(\delta + 1) > 1/\delta$, gRate sends dataset D_2 first. As both datasets would be sent over a loaded link for time δ if their transfers started at time 0, algorithm gSlowtime chooses the larger dataset, which is D_2 . Thus, both gRate and gSlowtime construct dataset sequence (D_2, D_1) , which gives a schedule of length $2\delta + 1$.

At moment 0, the time necessary to transfer D_1 is δ , and the time needed for transferring D_2 is $\delta + 1$. Hence, algorithm gTime constructs the optimum schedule (D_1, D_2) of length $\delta + 2 < 2\delta + 1$. $\square$

Proposition 11 *There exist instances which are solved to the optimum by gSlowtime, but not by gRate or gTime.*

Proof Let $m = 3$, $\alpha_1 = 2$, $\alpha_2 = 5$, $\alpha_3 = 4$, $\delta < 1.5$. Suppose that the first link is free only in interval $[0, 1)$, and the second link is free only in interval $[2\delta, 2\delta + 3)$. The third link is always loaded. If the respective transfers start at time 0, the average communication speeds for sending datasets D_1, D_2, D_3 will be $2/(\delta + 1)$, $5/(2\delta + 3)$, $1/\delta$, correspondingly. Hence, algorithm gRate will start with sending dataset D_2 in interval $[0, 2\delta + 3)$. After time $t = 2\delta + 3$ the links of P_1 and P_3 are always loaded. Hence, their communication speeds are the same, and gRate will first transfer D_3 , which is larger than D_1 , and finish with D_1 . Thus, the dataset sequence constructed by gRate is (D_2, D_3, D_1) , and the corresponding schedule length equals

$$T_{gRate} = 2\delta + 3 + 4\delta + 2\delta = 8\delta + 3. \quad (12)$$

If the respective transfers start at time 0, datasets D_1, D_2, D_3 will be sent in times $1 + \delta$, $2\delta + 3$, 4δ , correspondingly. The resulting slowdown times will be δ , 2δ , 4δ , respectively. Thus, algorithms gTime and gSlowtime send D_1 first. This transfer finishes at time $1 + \delta$. At this moment, the time required for the transfer of D_2 is $2\delta + 3$, and the transfer of D_3 would take time 4δ . Since $\delta < 1.5$, algorithm gTime selects D_3 , whose transfer is completed at time $1 + 5\delta > 2\delta + 3$. Thus, the whole dataset D_2 will be sent over a loaded link. The sequence constructed by gTime is (D_1, D_3, D_2) , which gives schedule length

$$T_{gTime} = 1 + \delta + 4\delta + 5\delta = 10\delta + 1. \quad (13)$$

After transferring dataset D_1 , algorithm gSlowtime has to choose between sending D_2 and D_3 , starting at time $\delta + 1$. The respective times of transfer over loaded links equal 2δ and 4δ . Hence, gSlowtime selects dataset D_2 . The obtained schedule is (D_1, D_2, D_3) , and its makespan is

$$T_{gSlowtime} = 1 + \delta + 3 + 2\delta + 4\delta = 7\delta + 4. \quad (14)$$

Fig. 3 Average execution time versus m . **a** $F = 1, L = 5$ and **b** $F = 5, L = 1$

Since $\delta > 1$, the schedule constructed by gSlowtime is shorter than the ones obtained by gTime and gRate. It is also easy to check that the solution delivered by gSlowtime is optimal. $\square$

7 Computational experiments

In this section, we compare experimentally the quality of delivered solutions and the computational costs of the proposed algorithms. The algorithms were implemented in C++ and run on an Intel Core i7-7820HK CPU @ 2.90 GHz with 32GB RAM.

We constructed sets of “small” instances with $m = 10, \dots, 25$, and “large” instances with $m = 10, 20, \dots, 100$. Parameter δ was set to 2. Dataset sizes α_i were selected randomly from interval $[1, 20]$. The communication speed changes were generated in the following way. The number of changes was controlled by parameters F and L , determining the maximum length of a free interval and the maximum length of a loaded interval, correspondingly. At time 0 all links were free. The length of the first free interval for link i was selected randomly from interval $[0, F \sum_{i=1}^m \alpha_i / m]$. Then, the length of the first loaded interval was chosen randomly from $[0, L \sum_{i=1}^m \alpha_i / m]$. We continued selecting the lengths of consecutive free and loaded intervals until reaching the time horizon $\bar{T}$. This procedure was repeated for each $i = 1, \dots, m$. We analyzed all combinations of $F, L \in \{1, \dots, 5\}$, but we will concentrate here only on the results obtained for the border values $F, L \in \{1, 5\}$. Note that when $F = 1$, the maximum possible length of a free interval equals the time required for sending an average-sized dataset over a free link. Thus, we will be forced to send at least some parts of most datasets over loaded links. Contrarily, when $F = 5$, we have a high chance of sending many datasets over free links only.

For each analyzed setting, 100 instances were generated and solved. For small instances, we measure the quality of heuristic solutions by the percentage error with respect to the optimum schedule length computed by DP. Due to the high complexity of DP algorithm, finding the optimum solutions was not possible for instances with $m \geq 30$. Therefore, for large instances, the quality of obtained schedules is measured by the percentage error with respect to the lower bound $\text{LoBo} = \sum_{i=1}^m \alpha_i$.

In Fig. 3 we present the average execution times of the algorithms for small instances. As expected, the running times of all algorithms increase with growing m , Rnd is the fastest algorithm, and DP is the slowest. Naturally, the impact of changing m is stronger for algorithms with higher computational complexity. The running times of all greedy heuristics are almost identical; hence, the lines corresponding to gTime, gRate and gSlowtime overlap in Fig. 3. The local search algorithms are significantly slower, and differences in their execution times are visible: gRateLocal is the fastest, and RndLocal is the slowest. Indeed, depending on the initial schedule, different numbers of local search steps may occur. By comparing Fig. 3a, b, we can see that the choice of F and L influences the running times of the local search algorithms. When $F = 1$ and $L = 5$ (Fig. 3a), the free intervals are short, and the loaded intervals are long. Hence, it is difficult to find a short schedule, and many iterations of local search are necessary. Contrarily, when $F = 5$ and $L = 1$ (Fig. 3b), the links are free for most of the time, and constructing a good schedule is much easier. Therefore, in this case all local search algorithms work faster than for $F = 1, L = 5$, especially if m is large. Although the choice of F and L affects the total number n of loaded intervals, it seems that its impact is too small to cause significant differences in the running times of the greedy heuristics, Rnd or DP.

The average quality of solutions delivered by the heuristics for instances with $F = 1$ is shown in Fig. 4. As algorithm

Fig. 4 Average solution quality versus m . **a** $F = L = 1$ and **b** $F = 1, L = 5$

DP always produces optimum solutions, we omit it for better clarity in all charts presenting schedule quality. It can be seen that the errors obtained for $L = 1$ (Fig. 4a) are smaller than in the case of $L = 5$ (Fig. 4b). Thus, we can say that the instances with big L are more difficult to solve. The general patterns are very similar in both cases. The solutions returned by Rnd become significantly worse when m increases. The impact of m on the remaining algorithms is much smaller. An increasing trend can be seen for the local search algorithms and gTime, but it is barely visible for gRate and gSlowtime. It seems that these algorithms are not very sensitive to changing m in the analyzed range. We conjecture that they retain similar quality for large instances. All greedy heuristics and local search algorithms perform much better than algorithm Rnd. The second worst algorithm is gTime. Indeed, this heuristic may prefer sending small datasets over loaded links to transferring big datasets over free links, thus obtaining long schedules. The results delivered by gSlowtime are much better than those of gTime, and gRate is the most effective of the simple heuristics. The best schedules are generated by local search algorithms. The differences between them are very small for $F = L = 1$, but they become larger when $L = 5$ and m is big. The average errors of all local search algorithms are below 6% when $L = 1$ and below 9% for $L = 5$. This means that local search is a very effective tool for solving our problem, and it can find good schedules even when starting from a random solution. Still, the best of local search algorithms is gRateLocal, whose average errors do not exceed 4.5% for $L = 1$ and 6.5% for $L = 5$. This conforms with gRate delivering the best results for the analyzed instances among simple heuristics. Moreover, by comparing Fig. 4b with Fig. 3a (both of which present the results obtained for $F = 1, L = 5$), we can see that the choice of the initial schedule affects the local search algorithms in two ways. Starting with a better schedule leads to finding a better final solution in a shorter time.

Figure 5 presents the quality of results obtained for $F = 5$. In Fig. 5a we can see that the errors obtained by all heuristics for $F = 5, L = 1$ are much smaller than in the remaining cases (Figs. 4, 5b). Indeed, when F is big and L is small, the links are very often free, and it is easy to find a short schedule. It is very interesting that the influence of m is different for big F than it was for small F . Increasing m leads now only to a slight degradation of Rnd performance. The quality of the solutions found by the remaining algorithms improves when m gets bigger. The local search algorithms are often able to find the optimum solutions for instances with $m \geq 20$. This can be explained by the fact that a larger number of nodes gives more opportunities to avoid a loaded link by sending datasets over other links, which are not loaded. When free intervals are long enough, this may lead to obtaining very good schedules. Another interesting observation is that for $F = 5$ algorithm gSlowtime outperforms gRate for all values of m when $L = 1$, and for some m when $L = 5$. Recall that algorithm gSlowtime concentrates on minimizing the length of data transfers over loaded links, while gRate aims at optimizing the average communication speed. Thus, the good results of gSlowtime confirm that when free intervals are long, the strategy of avoiding loaded intervals is advantageous.

To better analyze the impact of F and L on the obtained results, we conducted additional experiments for instances with $m = 20$. The range of F and L values was extended up to 10. In Fig. 6a we present the results obtained for $F = 1$ and variable L . It can be seen that the instances with larger L are harder to solve by most heuristics. However, when L reaches 5, the quality of solutions returned by all the algorithms seems to stabilize. Indeed, when L is very big, many loaded intervals have to be used in any schedule, and thus, the optimum solutions are longer. Hence, the maximum errors that can be obtained get smaller. This phenomenon is also confirmed by the Rnd solutions getting slightly better for

Fig. 5 Average solution quality versus m . **a** $F = 5$, $L = 1$ and **b** $F = L = 5$

Fig. 6 Average solution quality for $m = 20$. **a** $F = 1$, changing L and **b** $L = 5$, changing F

$L \geq 7$. The results obtained for $L = 5$ and variable F are shown in Fig. 6b. Increasing F generally leads to generating easier instances and, hence, getting better results. The only exception is that for $F = 1$ the errors of the heuristic solutions are smaller than for $F = 2$. This effect is similar to obtaining better results when L gets very large, which we discussed above. When free intervals are very short, then the optimum schedules are long, and in consequence, the relative errors of the heuristic solutions are smaller.

In the last set of experiments we analyze the algorithm performance for large instances with the most demanding combination of parameters $F = 1$, $L = 5$ (see Fig. 7). Let us remind that the schedule quality is now measured by the error with respect to the lower bound, and not the optimum solution, which is unknown. Thus, the errors reported in Fig. 7a are much larger than those presented in Figs. 4, 5, 6. The performance of Rnd is rather stable, while the results returned by the remaining algorithms get better when m becomes big, which stands in contrast to the results obtained for smaller m and $F = 1$, $L = 5$ (Fig. 4b). However, this effect can

be explained by the fact that the distance between the lower bound and the actual optimum becomes smaller for larger m . This claim is supported by the analysis of average relative distances between the optimum T^* and the lower bound LoBo computed for smaller instances, presented in Table 1. For each combination of F and L , the ratios T^*/LoBo decrease with increasing m . Thus, the trends visible in Fig. 7a reflect this behavior, rather than obtaining better approximations of the optimum solutions for very large m . It is interesting that for very big m , heuristic gRate outperforms local search algorithms other than gRateLocal. It seems that greedy optimization of the communication rates is a very good strategy for large instances. Figure 7b shows the running times of the heuristics for tests with big m . The differences between individual algorithms are clearer than for the small instances, but apart from that, the results are very similar to the ones presented in Fig. 3. The random algorithm is the fastest, and the running times of the greedy heuristics are again almost equal. (The corresponding lines overlap in Fig. 7b.) Choosing a good initial schedule results in a relatively small num-

Fig. 7 Results obtained for large instances with $F = 1$, $L = 5$, versus m . **a** Average solution quality and **b** average execution time

Table 1 Average ratios T^*/LoBo versus m

m	$F = 1,$ $L = 1$	$F = 1,$ $L = 5$	$F = 5,$ $L = 1$	$F = 5,$ $L = 5$
10	1.1328	1.3672	1.0003	1.0136
15	1.1027	1.3155	1.0000	1.0015
20	1.0868	1.2837	1.0000	1.0002
25	1.0763	1.2602	1.0000	1.0000

Table 2 The numbers and percentages of instances for which particular local search algorithms obtained the best solutions among all heuristics

Algorithm	Number of wins	Percentage of wins (%)
gTimeLocal	2562	27.26
gRateLocal	5469	58.18
gSlowtimeLocal	3420	36.38
RndLocal	3306	35.17

ber of iterations in local search algorithms and, hence, in a short execution time. In particular, the running time of gRateLocal is of the order of up to several seconds even for $m = 100$.

To summarize the comparison of local search algorithms, in Table 2 we present the numbers and percentages of instances for which each of them obtained the best result among all heuristics. This table covers all 9400 instances used in the experiments described above. The sum of percentages is greater than 100% because for some instances several algorithms delivered schedules of the same length. It can be seen that algorithm gRateLocal wins in most cases. Moreover, the previous experiments showed that in almost all test settings it achieves the best average solution quality among the analyzed heuristics, and that its execution time is quite low even for large instances. Thus, we recommend algorithm gRateLocal for solving our problem.

8 Conclusions

In this paper, we analyzed minimizing the time of gathering data in a star network whose performance is affected by background communications. We proved that although the preemptive version of this problem can be solved in polynomial time, the non-preemptive variant is strongly NP-hard. We proposed for it an exponential exact algorithm, a group of simple heuristics and a group of local search algorithms. It was shown that algorithms gTime, gRate and gSlowtime have tight approximation ratios equal to δ . We also proved that none of these three algorithms is dominated by a combination of the two remaining ones.

The performance of the algorithms was also tested in computational experiments. Algorithm DP cannot be used for large instances because of its high complexity. The quality of the heuristic solutions is influenced by parameters F , L , m . The simple heuristics have low runtime, but their results are often not very good. Among these algorithms, the best results are usually delivered by gRate, although we also found settings for which gSlowtime works better, and instances for which gTime obtains the best solutions. Local search algorithms offer a good compromise between quality and complexity. The choice of initial schedule influences the quality of the obtained schedules, as well as local search execution time. For the analyzed test instances, algorithm gRateLocal outperforms our remaining heuristics on average. It is worth noting that the proposed algorithms, with the exceptions of gSlowtime and gSlowtimeLocal, can also be used for solving a more general problem, where the communication speed of any link can change arbitrarily, instead of taking only two values.

Scheduling data gathering with variable communication speed is a new research area that contains many interesting problems which may be investigated in the future. Additional dataset parameters like priorities or due dates as well

as optimization criteria other than schedule length should be analyzed. Moreover, the network model used in this paper can be generalized in several different ways. Firstly, in some applications the collected data have to be further analyzed. Thus, the network works like a two-machine flow shop, as each dataset has to be first received and then processed by the base station. Secondly, the data can be collected and preprocessed by intermediate nodes before being passed to the base station. In such a case, the network has a two-level tree topology. Finally, in this work the single-port model was used, i.e., we assumed that the base station can receive only one dataset at a time. Under the bounded multi-port model, it is possible to have a limited number of parallel communications, which can make the scheduling problem even more challenging.

Acknowledgements The author is grateful to the Guest Editors and the anonymous Reviewers whose valuable comments helped improve the quality of this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*, 38, 393–422.
- Berlińska, J. (2014). Communication scheduling in data gathering networks with limited memory. *Applied Mathematics and Computation*, 235, 530–537.
- Berlińska, J. (2015). Scheduling for data gathering networks with data compression. *European Journal of Operational Research*, 246, 744–749.
- Berlińska, J. (2016). Scheduling data gathering with variable communication speed. In *Proceedings of the first international workshop on dynamic scheduling problems* (pp. 29–32).
- Berlińska, J. (2018a). Scheduling data gathering with maximum lateness objective. In R. Wyrzykowski, et al. (Eds.), *Parallel processing and applied mathematics: 12th International conference PPAM 2017, Part II, LNCS 10778* (pp. 135–144). Cham: Springer.
- Berlińska, J. (2018b). Scheduling non-preemptive data gathering affected by background communications. In *Proceedings of the second international workshop on dynamic scheduling problems* (pp. 41–44).
- Berlińska, J. (2020). Heuristics for scheduling data gathering with limited base station memory. *Annals of Operations Research*, 285, 149–159.
- Choi, K., & Robertazzi, T. G. (2008). Divisible load scheduling in wireless sensor networks with information utility. In *IEEE international performance computing and communications conference: IPCCC 2008* (pp. 9–17).
- Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman.
- Luo, W., Xu, Y., Gu, B., Tong, W., Goebel, R., & Lin, G. (2018a). Algorithms for communication scheduling in data gathering network with data compression. *Algorithmica*, 80(11), 3158–3176.
- Luo, W., Gu, B., & Lin, G. (2018b). Communication scheduling in data gathering networks of heterogeneous sensors with data compression: Algorithms and empirical experiments. *European Journal of Operational Research*, 271, 462–473.
- Moges, M., & Robertazzi, T. G. (2006). Wireless sensor networks: Scheduling for measurement and data reporting. *IEEE Transactions on Aerospace and Electronic Systems*, 42, 327–340.
- Szigeti, T., Hattingh, C., Barton, R., & Briley, K. (2013). *End-to-End QoS network design: Quality of Service for rich-media & cloud networks*. Indianapolis: Cisco Press.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.