Heuristics for scheduling data gathering with limited base station memory

Joanna Berlińska¹

Published online: 27 February 2019
© The Author(s) 2019

Abstract

In this paper, we analyze scheduling in data gathering networks with limited base station memory. The network nodes hold datasets that have to be gathered and processed by a single base station. A dataset transfer can only start if sufficient amount of memory is available at the base station. As soon as a node starts sending a dataset, the base station allocates a block of memory of corresponding size. The memory is released when computations on the dataset finish. We prove that minimizing the total data gathering and processing time is strongly NP-hard. As this problem is a special case of a specific resource constrained flow shop scheduling problem, for which an exact exponential algorithm is known, we propose several simple polynomial-time heuristics and two groups of local search algorithms, and test their performance in computational experiments. We show that the local search algorithms produce very good schedules, and one of the simple heuristics delivers solutions of comparable quality in a very short time.

Keywords Scheduling · Data gathering network · Limited memory · Flow shop

1 Introduction

Data gathering networks find a wide range of applications. Many computational tasks can be divided between a set of computers running in parallel. Each of such workers produces some results, and all these data have to be collected together on a single machine, for aggregating, further processing and storing. Moreover, data gathering wireless sensor networks are used in environmental, military, health and home applications (Akyildiz et al. 2002). The efficiency of collecting the data has an impact on the performance of the whole distributed application. Therefore, scheduling for data gathering networks is an important research area.

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 in Poznań, Umultowska 87, 61-614 Poznań, Poland

Moges and Robertazzi (2006) and Choi and Robertazzi (2008) studied data gathering networks on the grounds of divisible load theory. The analyzed problem was to assign the amounts of measured data to the network nodes and organize the communications in the network so as to minimize the total time of sensing and gathering the data. Later on, scheduling algorithms for networks with fixed sizes of data gathered by individual nodes were proposed. The analyzed objectives included maximizing the network lifetime (Berlińska 2014), minimizing the time of data gathering (Berlińska 2015; Luo et al. 2018a,b), and minimizing the maximum lateness (Berlińska 2018a).

In this paper, we analyze gathering data in networks with limited base station memory. Each worker holds a dataset that should be passed to the base station for processing. A dataset being transferred to, or processed by the base station, occupies a block of memory of a given size. The total size of coexisting memory blocks cannot exceed the base station buffer capacity. Our goal is to gather and process all data within the minimum possible time. We prove that this problem is strongly NP-hard. Then, we propose a group of polynomial-time heuristics and local search algorithms. Their performance and sensitivity to changes of instance parameters are tested in a series of computational experiments.

The rest of this paper is organized as follows. In Sect. 2 we describe the network model and formulate the scheduling problem. In Sect. 3 related work is outlined. The computational complexity of our problem is analyzed in Sect. 4. Heuristic algorithms are proposed in Sect. 5, and the results of computational experiments on their performance are presented in Sect. 6. The last section is dedicated to conclusions.

2 Problem formulation

We study a data gathering network consisting of m worker nodes $P_1, \dots, P_m$, and a single base station. Node P_i has to transfer dataset D_i of size α_i directly to the base station. When dataset D_i starts being sent, a memory block of size α_i is allocated at the base station. The base station has limited memory of size $B \geq \max_{i=1}^m \{\alpha_i\}$. The transfer of dataset D_i may start only if the amount of available memory is at least α_i . Sending dataset D_i requires time $C\alpha_i$. After dataset D_i is transferred, it has to be processed by the base station, which takes time $A\alpha_i$. Datasets are processed in the order in which they were received, without unnecessary delay. As soon as processing a dataset finishes, the corresponding memory block is released. Both communication and computation on a dataset are non-preemptive. The base station can communicate with at most one node at a time, and it can process at most one dataset at a time. The scheduling problem is to choose a sequence of dataset transfers such that the total data gathering and processing time is minimized.

Note that since at most one dataset can be transferred at a time, our data gathering network is a two-machine permutation flow shop, where the communication network is the first machine, and the base station is the second machine. For each dataset D_i , we have a corresponding job i consisting of two operations: sending dataset D_i in time $p_{1i} = C\alpha_i$, and processing this dataset in time $p_{2i} = A\alpha_i$. Executing job i requires α_i units of the base station memory buffer.

3 Related work

Most research on two-machine flow shop scheduling with limited buffer considers the case when a limited number of jobs can be held in a storage buffer after being processed on the first

machine and before being started on the second one. For the case when no job can be stored in a buffer (called the no-wait problem), a polynomial-time algorithm proposed by Gilmore and Gomory (1964) for solving a specific TSP can be used to obtain optimum solutions. The case with unlimited buffer can also be solved in polynomial time, using Johnson's rule (Johnson 1954). The remaining case of positive, finite buffer size, was shown to be strongly NP-hard by Papadimitriou and Kanellakis (1980).

The problem analyzed in this paper significantly differs from the one described above. Firstly, in our case the base station buffer can hold a fixed amount of data rather than a fixed number of datasets. For example, in a buffer of size 4, we can store only one dataset of size 4, but up to 4 datasets of size 1. Secondly, the buffer is occupied by a dataset not only between, but also during the two operations of its transfer and processing.

A flow shop with such a quantity-based buffer was first studied by Lin et al. (2009). The analyzed problem was to optimize the object sequence in a prefetch-enabled TV-like presentation. It was assumed that the execution time of the first operation of a job is proportional to its buffer requirement, but the execution time of the second operation may be arbitrary. Minimizing the schedule length was proved to be strongly NP-hard, and a branch and bound algorithm was proposed. The performance of this algorithm was further improved by adding new lower bounds by Kononov et al. (2012). Integer linear programming formulations and variable neighborhood search algorithms were proposed by Kononova and Kochetov (2013). The existence of optimal permutation schedules for this problem was analyzed by Fung and Zinder (2016).

Lin and Huang (2006) analyzed a relocation problem with a second working crew for resource recycling. In their work, each job was executed on two machines in a permutation flow shop style. The execution time of job i on the first machine was denoted by p_i , and on the second machine by q_i . Job i required α_i units of a resource, and returned β_i units of this resource on completion. The goal was to minimize the makespan while not exceeding the available amount of the resource. This problem, denoted as $F2|rp|C_{max}$, was shown to be strongly NP-hard, and heuristic algorithms for solving it were proposed. The problem was further analyzed by Cheng et al. (2012), who formulated it as an integer linear program. Complexity results for a number of special cases were presented. The authors also studied the non-permutation version of the problem. Since in our problem, job i requires α_i memory and returns the same amount of memory after completion, we solve a special case of the permutation version of $F2|rp|C_{max}$, which can be denoted as $F2|rp, p_i = C\alpha_i, q_i = A\alpha_i, \beta_i = \alpha_i|C_{max}$.

In our preliminary work on the problem studied here (Berlińska 2018b), we proposed several heuristics and tested the quality of delivered solutions in computational experiments. In this paper, we design even more algorithms and present their experimental comparison for a wider variety of instance parameters.

4 Computational complexity

In this section, we show that although our problem is a simpler special case of the problems analyzed by Lin and Huang (2006) and Lin et al. (2009), it remains strongly NP-hard. To this end, we use a reduction from the following strongly NP-complete BIN- PACKING problem (Garey and Johnson 1979).

Bin-Packing: Given positive integers V and k , and a set of n positive integers $\{x_1, \dots, x_n\}$, is it possible to partition the index set $\mathcal{N} = \{1, 2, \dots, n\}$ into k disjoint sets $N_1, N_2, \dots, N_k$, such that for each $1 \leq j \leq k$, $\sum_{i \in N_j} x_i \leq V$?

Proposition 1 *Makespan minimization in data gathering networks with limited base station memory is strongly NP-hard even if $A = C = 1$.*

Proof It is clear that the decision version of our problem belongs to NP. We perform a pseudo-polynomial reduction from the Bin-Packing problem. Given an instance of Bin-Packing, we construct the following instance of our scheduling problem. The network has $m = n + k + 1$ workers, memory of size $B = 2V + 1$, and $A = C = 1$. There are n ordinary datasets $D_1, \dots, D_n$ of size $\alpha_i = x_i$ for $i = 1, \dots, n$, and $k + 1$ enforcer datasets $D_{n+1}, \dots, D_{n+k+1}$ of size $\alpha_i = V + 1$ for $i = n + 1, \dots, n + k + 1$. We will show that a schedule not longer than $T = 2(k + 1)(V + 1)$ exists if and only if the corresponding instance of Bin-Packing is a “yes”-instance.

Let us first assume that a schedule of length at most T exists. Note that the base station buffer cannot hold two or more enforcer datasets at a time, since $2(V + 1) > B$. Thus, all enforcer datasets have to be sent and processed in disjoint time intervals. Therefore, the time required for transferring and processing all enforcer datasets is at least $(C + A)(k + 1)(V + 1) = T$. Hence, the enforcer datasets have to be transferred in intervals $[2i(V + 1), (2i + 1)(V + 1))$, and processed in intervals $[(2i + 1)(V + 1), (2i + 2)(V + 1))$, for $i = 0, \dots, k$ (see Fig. 1). Since no dataset is ready for processing in interval $[0, V + 1)$, all ordinary datasets have to be processed in k disjoint intervals $[2j(V + 1), (2j + 1)(V + 1))$, for $j = 1, \dots, k$ (cf. Fig. 1). All the ordinary datasets that will be processed in a given interval $[2j(V + 1), (2j + 1)(V + 1))$ have to be sent before time $2j(V + 1)$, when the transfer of an enforcer dataset starts. Hence, at time $2j(V + 1)$ all the ordinary datasets assigned to the analyzed interval coexist in the base station memory with the block allocated for the enforcer dataset. Thus, their total size cannot exceed $B - (V + 1) = V$. Therefore, the sizes x_i of the ordinary datasets processed in interval $[2j(V + 1), (2j + 1)(V + 1))$ fit together in a bin of size V , for each $j = 1, \dots, k$. Hence, we have the required feasible solution to Bin-Packing problem.

Conversely, assume that the analyzed instance of Bin-Packing is a “yes”-instance. We construct a schedule in the following way (see Fig. 1). Enforcer dataset D_{n+1+i} is sent in interval $[2i(V + 1), (2i + 1)(V + 1))$, and processed in interval $[(2i + 1)(V + 1), (2i + 2)(V + 1))$, for $i = 0, \dots, k$. The ordinary datasets corresponding to numbers x_i packed in the j th bin, are sent one by one in interval $[(2j - 1)(V + 1), 2j(V + 1))$, and they are processed sequentially in interval $[2j(V + 1), (2j + 1)(V + 1))$, for $j = 1, \dots, k$. It is easy to see that the memory limit is observed, and the schedule length equals T .

All in all, we showed that a feasible solution of a given instance of Bin-Packing exists if and only if there exists a schedule with makespan not greater than T for the corresponding instance of our scheduling problem, which completes the proof. $\square$

Fig. 1 Schedule structure for the Proof of Proposition 1. Enforcer datasets are dark gray, ordinary datasets are light gray

To conclude this section, we indicate two special cases of our problem that can be easily solved in polynomial time. If $B < \min_{i \neq j} \{\alpha_i + \alpha_j\}$, then no two datasets can overlap in the base station memory, and each dataset permutation yields an optimum schedule of length $(C + A) \sum_{i=1}^m \alpha_i$.

If B is large enough (for example, $B \geq \sum_{i=1}^m \alpha_i$), then the optimum schedule is constructed in $O(m \log m)$ time by Johnson's algorithm (Johnson 1954). Note that the above constraint on B is very rough. In many cases it is enough that $B \geq \max_{i \neq j} \{\alpha_i + \alpha_j\}$. In order to check if B is large enough, we can compute the schedule length for the dataset permutation returned by Johnson's algorithm and compare it with the makespan obtained for the same sequence and an infinite base station buffer. If these two makespans are equal, then it is certain that the optimum schedule has been found.

5 Heuristic algorithms

Before we start describing the proposed algorithms, let us observe the following symmetry property. Suppose that $A = kC$, where $k \geq 1$, and Σ is a schedule of length T for given values of B and $(\alpha_i)_{i=1}^m$. Then, by reversing schedule Σ and swapping communications with computations, we obtain a schedule of length T for the same values of B and $(\alpha_i)_{i=1}^m$, computation rate $A' = C$, and communication rate $C' = A = kA'$. Therefore, from now on we will assume without loss of generality that $A \geq C$.

As our problem is a special case of the permutation version of $F2|rp|C_{max}$, we do not propose an exact exponential algorithm, since the ILP formulation given by Cheng et al. (2012) can be used to find optimum schedules. However, this approach is not practical because of its high computational complexity. Therefore, we construct fast heuristic algorithms, and analyze the quality of delivered solutions.

We start with a group of "simple" heuristics, each of which constructs a single dataset sequence in $O(m \log m)$ time. Algorithm **Inc** sorts the datasets in the order of increasing sizes. Note that since we assumed $A \geq C$, such a dataset sequence would be returned by Johnson's algorithm, and hence, algorithm **Inc** delivers optimum solutions if the memory limit B is big enough. Algorithm **Alter** starts with sending the smallest dataset, then the greatest one, the second smallest, the second greatest, etc., thus alternating big and small datasets. The idea behind this approach is to avoid sending the biggest datasets one after another. Indeed, if the base station memory is not very large, big datasets will not fit in the buffer together, and hence, idle times due to waiting for memory release will occur. Therefore, we aim here at forming pairs of consecutively sent datasets, such that the total size of each pair is not too big and fits in a medium-sized buffer. Algorithm **LF** constructs a schedule step by step, always choosing to send the largest dataset that fits in currently available memory. If all datasets are too big, the communication network is idle until sufficient amount of memory is released, such that some dataset can be transferred. Finally, algorithm **Rnd** constructs a random dataset sequence. This algorithm is used mainly to verify if the remaining heuristics perform well in comparison to what can be achieved without effort.

The second group of proposed heuristics are 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 if 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 is continued until no further improvement is possible. These algorithms are called **IncSwap**,

AlterSwap, **LFSwap** and **RndSwap**. The beginning of each name shows which simple heuristic is used to construct the initial schedule.

Local search algorithms based on swaps were used for solving problem $F2|rp|C_{max}$ by Lin and Huang (2006), and they were shown to deliver good solutions. However, using a different neighborhood in a local search algorithm may result in obtaining even better results. Therefore, we also analyze local search algorithms based on dataset shifts. Here, instead of swapping a pair of datasets, we move a single dataset into a different position in the schedule. Reflecting the method used for generating the initial schedule, these algorithms are called **IncShift**, **AlterShift**, **LFSHift** and **RndShift**.

Lin and Huang (2006) proposed three heuristics H_1 , H_2 , H_3 for solving $F2|rp|C_{max}$. However, for the special case of this problem analyzed in our work, both H_1 and H_2 yield the same results as IncSwap, and H_3 is equivalent to RndSwap. Thus, it is not necessary to additionally include algorithms H_1 , H_2 and H_3 in our study.

To finish this section, let us note that regardless of the selected dataset sequence, the schedule length never exceeds $(A + C) \sum_{i=1}^m \alpha_i$. Moreover, the total computation time $A \sum_{i=1}^m \alpha_i$ is a lower bound on the makespan. Thus, the approximation ratio of any algorithm for solving our problem is at most $1 + C/A$. Hence, we can say that our problem is easier to solve when A is large in comparison to C .

6 Experimental results

In this section, we compare the quality of delivered solutions and the computational costs of the proposed heuristics. The algorithms were implemented in C++ and run on an Intel Core i7-7820HK CPU @ 2.90 GHz with 32GB RAM. Integer linear programs were solved using Gurobi (Gurobi Optimization 2016). The test instances were constructed as follows. The network parameters were $C = 1$ and $A \in \{1, 2\}$. We used only two values of parameter A , because, as we explained at the end of Sect. 5, instances with $A \gg C$ are less demanding. We generated “small” tests with $m = 10$ and “big” instances with $m = 100$. Dataset sizes α_i were chosen randomly from the interval $[1, 1 + \delta_\alpha]$, where $\delta_\alpha \in \{0.5, 1, 1.5, 2\}$. Note that if δ_α is very small, then all datasets have similar sizes, and in consequence, the differences between makespans obtained for various dataset sequences are also small. For a given set of sizes α_i , we computed the minimum amount of memory that allows to hold more than one dataset in the buffer, $B_{min} = \min_{i \neq j} \{\alpha_i + \alpha_j\}$. Then, the memory limit was set to $B = \delta_B B_{min}$, where $\delta_B = 1 + i/10$, for $i = 1, 2, \dots, 8$.

The quality of schedules constructed for small instances was measured by the ratio T/OPT , where T is the makespan obtained by a given heuristic, and OPT is the optimum schedule length delivered by the ILP formulation proposed by Cheng et al. (2012) for the permutation version of $F2|rp|C_{max}$. Finding optimum solutions for big instances was not possible because of the exponential complexity of the exact algorithm. Therefore, we computed a lower bound $LoBo$ on the schedule length, by disregarding the memory limit and solving the resulting instance of problem $F2||C_{max}$ using Johnson’s rule. Thus, the measure of schedule quality for the big instances is $T/LoBo$. Each point on the following charts represents an average over 100 instances.

In the first set of experiments, we analyze the influence of the size of available memory on the obtained solutions, for small instances with $A = 1$ and $\delta_\alpha = 1$. The simple heuristics Inc, Alter and Rnd deliver the worst results (see Fig. 2a). Still, it is interesting to identify the factors determining the performance of Inc and Alter. When the memory limit is very small,

Fig. 2 Solution quality vs. δ_B for $m = 10$, $A = 1$, $\delta_\alpha = 1$. **a** All algorithms, **b** only local search algorithms

algorithm Inc performs well, because it places the smallest datasets together, so that they can overlap in the base station memory. Algorithm Alter performs worse even than Rnd, because a pair of a big and a small dataset does not fit in the buffer. When $\delta_B \in [1.4, 1.6]$, algorithm Inc performs badly. Indeed, as big datasets are sent one after another, they have to be transferred and processed separately, instead of overlapping with smaller datasets. Algorithm Alter can now create a lot of pairs of overlapping datasets, and hence, its performance is better. When the buffer becomes very big ($\delta_B \geq 1.7$), idle times due to waiting for a memory release are rarely necessary, and schedule length can often be minimized using Johnson's rule. Therefore, Inc obtains good results again, while Alter loses performance. It can be seen that algorithm LF performs best of all simple heuristics. The schedules it constructs are almost as good as the ones found by the local search algorithms. The maximum average error of LF (reached for $\delta_B = 1.3$) is below 3%. When the memory limit is big enough ($\delta_B \geq 1.6$), LF almost always finds optimum solutions.

Very good results are obtained by the local search algorithms. A closer look on their performance is presented in Fig. 2b. It can be seen that the instances with very small or very big memory limit are easiest to solve. It is worth noting that the algorithms based on shifting datasets in the sequence perform slightly better than their counterparts using swaps. However, as in most cases all local search algorithms deliver solutions at most 1% longer than the optimum, the differences are not very significant. The best results for this test set were produced by algorithm IncShift, which found optimum schedules for all 800 instances.

The results obtained for $m = 10$, $A = 1$ and $\delta_\alpha = 2$ are shown in Fig. 3. Bigger δ_α means that the dataset sizes are more diversified, and larger datasets appear than for $\delta_\alpha = 1$. Thus, increasing δ_B by 0.1 results in a smaller change of the buffer size in relation to the largest dataset size. As a result, algorithm Alter now reaches its worst point at $\delta_B = 1.3$ instead of 1.2, and it is not yet outperformed by Inc at $\delta_B = 1.7, 1.8$. The schedules delivered by LF are still very good, with the maximum average error below 3.5%. Among the local search algorithms, the best results are again obtained by IncShift, although it does not always find optimum solutions for instances with big δ_B . For $\delta_B \in [1.5, 1.7]$, algorithm IncShift is even outperformed by LFSHift.

The effects of increasing A to 2 (for $m = 10$, $\delta_\alpha = 1$) are presented in Fig. 4. It can be seen that the results of all heuristics improve in comparison to the case of $A = 1$ (Fig. 2). This is caused by decreasing the upper bound $1 + C/A$ on the schedule quality. The instances with $\delta_B \geq 1.5$ are now particularly easy to solve for all algorithms except Inc and

Fig. 3 Solution quality vs. δ_B for $m = 10$, $A = 1$, $\delta_\alpha = 2$. **a** All algorithms, **b** only local search algorithms

Fig. 4 Solution quality vs. δ_B for $m = 10$, $A = 2$, $\delta_\alpha = 1$. **a** All algorithms, **b** only local search algorithms

Rnd. The relationships between the results delivered by different algorithms are similar to the ones observed in the previous experiments. The best results overall are again achieved by IncShift.

Figure 5 shows the quality of solutions obtained for big instances. Let us remind that the measure of schedule quality is now the relative distance from the lower bound rather than from the optimum solution. When the base station buffer is big, the lower bound is close to the actual optimum, but when the memory is small, the distance between them may grow up to $1 + C/A$. Hence comes the slope of lines in Fig. 5. Taking that into account, the results are similar to the ones obtained for the small instances. By comparing Fig. 5a, b, where $A = 1$ and $A = 2$, respectively, we confirm that although the value of A determines the maximum possible errors of the heuristics, it has almost no influence on the relationships between the individual algorithms.

In Fig. 6 we present the trade-off between schedule quality and algorithm execution time for all tests with $m = 100$ and $A = 1$. Here, each point represents an average over 3200 instances. All the simple heuristics are very fast, although it seems that Alter is slower than the other ones. Algorithms Inc, Alter and Rnd produce results of similar average quality. This is explained by the fact that each of heuristics Inc and Alter performs much better

Fig. 5 Solution quality vs. δ_B for $m = 100$, $\delta_\alpha = 1$. **a** $A = 1$, **b** $A = 2$

Fig. 6 Solution quality vs. execution time for $m = 100$, $A = 1$

than Rnd on instances with some values of δ_B , but is counterproductive (i.e., achieves worse results than Rnd) on the remaining tests. The results returned by heuristic LF are as good as those delivered by the local search algorithms. The average quality of LF schedules is even a little better than those of both variants of local search starting with Rnd or Alter sequence. Local search algorithms based on shifts are slower than their counterparts using swaps. The running time of local search also depends on the initial schedule. Algorithms starting with the LF schedules are the fastest, and the order of the remaining algorithms is: Inc, Alter, Rnd. Small differences in the quality of the local search algorithms are also visible. The best results are found by starting with LF or Inc schedules. Using Alter for generating the initial solution does not yield such good results, and Rnd is even worse. Thus, choosing a bad initial sequence results both in longer execution time and in worse solutions found. Still, the differences between the average results obtained by LF and all local search algorithms are very small. As LF is several orders of magnitude faster than local search, we recommend it as the best heuristic in the terms of trade-off between quality and time.

7 Conclusions

In this work, we analyzed scheduling data gathering with limited base station memory. We proved that this problem is strongly NP-hard. Simple heuristics and local search algorithms

were proposed, and their performance was tested in computational experiments. We showed that the amount of available memory is the key parameter determining the performance of the heuristics. In general, it is easier to find good solutions if the memory limit is very small or very large. However, algorithm Alter performs best for the medium values of buffer size. Increasing the time A of processing one unit of data causes all algorithms to return better results, as the upper bound on the schedule quality $1 + C/A$ becomes smaller. Changing the dispersion of dataset sizes does not significantly influence the performance of the heuristics.

The average quality of results delivered by algorithms Inc and Alter is close to that of the random algorithm Rnd. Thus, these heuristics should not be used for solving our problem. Contrarily, algorithm LF produces very good schedules. Local search turned out to be a very effective way of constructing good solutions. It allows to find high quality schedules even if we choose a bad initial dataset sequence. Local search using dataset shifts is slower than that based on swaps, but it generates slightly better solutions. Still, the results obtained by the simple LF heuristic are close to those delivered by local search algorithms, and the running time of LF is much shorter. Thus, LF offers the best trade-off between the quality of results and the algorithm execution time, and is recommended as a very good tool for solving our problem. In the case when time is not an important factor, algorithm IncShift should be used, since its average results are the best among all analyzed heuristics, and it very often finds optimum schedules for the small instances.

In the future, the existence of theoretical guarantees (tighter than $1 + C/A$) on the approximation ratios of the presented heuristics should be investigated. A possible extension of this work is to analyze data gathering networks in which the size of available base station memory varies in time. Such changes may be caused by other applications and services running on the base station. Constructing good scheduling algorithms for such systems will be an interesting challenge.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

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. (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 data gathering with limited base station memory. In *Proceedings of the 16th international conference on project management and scheduling, Rome 2018* (pp. 42–45).
- Cheng, T. C. E., Lin, B. M. T., & Huang, H. L. (2012). Resource-constrained flowshop scheduling with separate resource recycling operations. *Computers & Operations Research*, 39, 1206–1212.
- Choi, K. & Robertazzi, T.G. (2008). Divisible load scheduling in wireless sensor networks with information utility. In *IEEE international performance computing and communications conference 2008: IPCCC 2008* (pp. 9–17).
- Fung, J., & Zinder, Y. (2016). Permutation schedules for a two-machine flow shop with storage. *Operations Research Letters*, 44(2), 153–157.

- Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman.
- Gilmore, P. C., & Gomory, R. E. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5), 655–679.
- Gurobi Optimization, Inc. (2016). Gurobi Optimizer Reference Manual. <http://www.gurobi.com>.
- Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61–68.
- Kononov, A., Hong, J. S., Kononova, P., & Lin, F. C. (2012). Quantity-based buffer-constrained two-machine flowshop problem: Active and passive prefetch models for multimedia applications. *Journal of Scheduling*, 15(4), 487–497.
- Kononova, P., & Kochetov, Y. (2013). The variable neighborhood search for the two machine flow shop problem with a passive prefetch. *Journal of Applied and Industrial Mathematics*, 7(1), 54–67.
- Lin, F. C., Hong, J. S., & Lin, B. M. (2009). A two-machine flow-shop problem with processing time-dependent buffer constraints: An application in multimedia presentations. *Computers & Operations Research*, 36(4), 1158–1175.
- Lin, B. M. T., & Huang, H. L. (2006). On the relocation problem with a second working crew for resource recycling. *International Journal of Systems Science*, 37, 27–34.
- 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.
- Papadimitriou, C. H., & Kanellakis, P. C. (1980). Flowshop scheduling with limited temporary storage. *Journal of the Association for Computing Machinery*, 27(3), 533–549.

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