4OR (2021) 19:265–288
https://doi.org/10.1007/s10288-020-00449-8

RESEARCH PAPER

Dynamic programming algorithms and Lagrangian lower bounds for a discrete lot streaming problem in a two-machine flow shop

Arianna Alfieri[1] · Shuyu Zhou[2,3] · Rosario Scatamacchia[1] · Steef L. van de Velde[3]

Received: 13 August 2019 / Revised: 29 May 2020 / Published online: 10 July 2020
© Springer-Verlag GmbH Germany, part of Springer Nature 2020

Abstract
In this paper, we propose exact and heuristic solution approaches based on dynamic programming for an open lot streaming problem. We also present the first application of Lagrangian relaxation to compute strong lower bounds to such a problem. The application concerns the minimization of the total flow time for the discrete version of a single-job lot streaming problem from the literature in a two-machine flow shop with attached setup times. Computational results on benchmark instances illustrate the effectiveness of the proposed approaches and give evidence of the strength of the Lagrangian relaxation lower bounds.

Keywords Lot streaming · Setup times · Flow shop · Total flow time · Lagrangian relaxation · Dynamic programming

Mathematics Subject Classification 90-08 · 90B35 · 90C39

✉ Arianna Alfieri
arianna.alfieri@polito.it

Shuyu Zhou
zshuyu@rsm.nl

Rosario Scatamacchia
rosario.scatamacchia@polito.it

Steef L. van de Velde
svelde@rsm.nl

1 DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

2 East China University of Science and Technology, No. 130, Meilong Rd, Shanghai, China

3 Rotterdam School of Management, Erasmus University, P.O. Box 1738, 3000, DR Rotterdam, The Netherlands

1 Introduction

Lot streaming is a relatively simple but effective scheduling technique to reduce manufacturing lead times. The idea is to split up a large lot of n identical items into sublots, which can be independently released to the next manufacturing stage once they have finished on the current stage, without waiting for the completion of the items belonging to the other sublots. Lot streaming allows the overlapping of different sublots processing at different stages, thus enabling the items to flow faster through the system. This eventually results in shorter overall flow times and faster customer deliveries (Santos and Magazine 1985; Silver et al. 1998). The lot streaming problem is similar to the batching problem typically arising in single machine environments where, however, also scheduling aspects may be present (Mosheiov et al. 2004; Agnetis et al. 2009).

Comprehensive reviews on lot streaming problems can be found in Chang and Chiu (2005) and Cheng et al. (2013) from which it can be observed that most of the lot streaming research deals with makespan minimization in flow shops with one machine per manufacturing stage.

In the case of makespan minimization, both the discrete variant, in which the sublot sizes need to be integral, and the continuous variant, in which the sublot sizes may assume fractional values, have been extensively studied (Trietsch 1987; Potts and Baker 1989; Trietsch and Baker 1993; Glass et al. 1994; Glass and Potts 1998; Cheng et al. 2000). The study of the continuous variants of lot streaming problems is motivated by the assumption that a good approximate solution for the discrete variant can be found by rounding an optimal solution for the continuous variant if the number of items n is large enough (Trietsch and Baker 1993; Chen and Steiner 1999; Bukchin et al. 2002).

It could seem obvious that the greater the number of sublots into which a lot is divided, the larger the overlapping is among the operations at the different stages of the production system. However, when setup times are paid at the beginning of each sublot, the greater the number of sublots into which a lot is divided, the larger the total setup time is and then process overlapping and total setup time must be traded-off to find the optimal number and sizes of sublots.

Setup times required before the beginning of each sublot are usually present in manufacturing environments such as semi-conductor industry (Kalir and Sarin 2003) and discrete parts manufacturing (Cheng et al. 1996). In these environments, sublots need to be loaded on machines and this activity can take a non-negligible time and needs the machines to be idle (Bukchin et al. 2002; Bukchin and Masin 2004; Chen and Steiner 1996, 1998; Ben-Dati et al. 2009).

When the machine setup can be performed only when all units of the sublot are available, it is called *attached* (or *non-anticipatory*). Although results for lot streaming problems with setup times are mainly found for the case of two-machine flow shops, also other environments have been considered such as hybrid flow shops (Xuan and Tang 2007; Cheng et al. 2016), three-machine flow shops (Chen and Steiner 1998) and job shops (Defersha and Chen 2012).

In the case of equal sublot sizes, closed formulae for the optimal number of sublots and for the optimal sublot sizes have been devised both for total flow time and makespan minimization (Kalir and Sarin 2001, 2003). In the case of single-lot, multiple-machine flow shop and unified cost-based objective function (including criteria pertaining to

makespan, mean flow time, work-in-process, sublot-attached setup and transfer times) Kalir and Sarin (2008) propose a polynomial-time procedure for determining the number of sublots resulting in near-optimal solution.

The case in which sublots may vary their size over the stages is studied in Biskup and Feldmann (2006), by using a mixed integer programming formulation, for a different settings and objective functions.

When makespan minimization and consistent sublot sizes (i.e., equal sublot sizes on each machine) are considered in a two-stage flow shop, Alfieri et al. (2012) prove that the continuous variant of the problem is solvable in $O(n)$ time, whereas the discrete variant is solvable in $O(n^3)$ time by a dynamic programming algorithm. Alfieri et al. (2012) also show that, in the case of makespan minimization, simple rounding strategies have both a compelling worst-case and empirical performance.

In this paper, we address the lot streaming problem introduced by Bukchin et al. (2002) of minimizing the total flow time in a two-stage flow shop with consistent sublot sizes. No optimization algorithm is known for this problem, neither for the continuous, nor for the discrete version. Also, no complexity results are known, i.e., to the best of our knowledge it is an *open* problem, as is the case of many lot sizing and lot streaming problems.

Bukchin et al. (2002) address the continuous variant of the problem, and they present an efficient heuristic procedure as well as an analytical lower bound, which by definition is also a lower bound on the optimal solution value for the discrete variant.

The contribution of our work is twofold. First, we propose exact and heuristic solution approaches for the discrete variant of the problem based on dynamic programming. In particular, a heuristic dynamic programmig algorithm combined with a local search procedure proves to be very effective for solving benchmark instances from the literature. Second, we derive strong lower bounds for the problem that rely on Lagrangian relaxation and that can be used to certify the quality of the solutions provided by heuristic approaches.

Whereas Lagrangian relaxation has shown its merit for many combinatorial optimization problems (Fisher 1985), we are not aware of any application to lot streaming problems. Hence, a further contribution of our paper is the first application of Lagrangian relaxation to a lot streaming problem, and as our Lagrangian approach is generic, we hope that other approximate approaches for lot streaming problems may follow suit.

The plan of the paper is as follows. In Sect. 2, we formally introduce the problem and derive a number of properties possessed by a class of optimal schedules. The dynamic programming algorithms are presented in Sect. 3. In Sect. 4, we give a mathematical formulation of the problem restricted to the class of optimal schedules defined in Sect. 2. We then present two different Lagrangian relaxations and upper bounds for the number of items in each sublot in the optimal solution ("Appendix A"). Computational results are presented in Sect. 5. Section 6 concludes the paper with some remarks.

2 Problem description and basic solution properties

We consider a two-machine flow shop in which n identical items are to be processed. Each item has to visit machine M_1 and machine M_2 in exactly this same order. The required processing times on M_1 and M_2 are p_1 and p_2, respectively. The objective is to partition the n identical items into b sublots, so as to minimize the total flow time (i.e., the sum of the completion times of the items on M_2). Since the total number of items n is given, the minimum total flow time solution corresponds to the minimum average flow time solution. The number of sublots b, instead, is not a-priori given, i.e., a solution of the problem has to provide both the number of sublots and their size.

Sublot sizes need to be integral, strictly positive, and *consistent*, i.e., the ith sublot in the schedule has to have the same size on both machines M_1 and M_2, for any $i \geq 1$. Furthermore, the size of the ith sublot does not have to be equal to the size of the $(i+1)$th sublot, for any $i \geq 1$; in other words, the sublot sizes are *general*. We assume sublot availability, which means that all items in the same sublot enter and leave each machine at the same time; accordingly, an item belonging to sublot i can be processed on M_2 only if all items in sublot i have been processed on M_1. In addition, an attached setup time is required on each machine before the items in a sublot can be processed. We let s_1 and s_2 denote these setup times on M_1 and M_2, respectively.

This problem occurs in manufacturing environments such semiconductor manufacturing and discrete part shops. In these environments, items of the same type are mounted on pallets and machines need to calibrate before a pallet can be worked on. Moreover, pallets need to be loaded onto machines and this can take a non-negligible time, usually larger than the time to process a single item. We refer the reader to the work of Bukchin et al. (2002) for a more in-depth discussion of the problem setting and industrial examples.

Since all items are identical and sublots are consistent, we may restrict the search for an optimal solution to a single set of size sequences that will be identical on each machine. Due to the similarity with the concept of permutation schedule in flow shop scheduling, we call such size sequence *permutation schedule*. Any permutation schedule σ can be represented as $\sigma = (x_1, \ldots, x_b)$, where x_i denotes the size of sublot i $(i = 1, \ldots, b)$ and $\sum_{i=1}^{b} x_i = n$. There always exists an optimal permutation schedule σ with the following properties:

1. $x_i \geq 1$ for $i = 1, \ldots, b$;
2. M_1 processes all sublots without any idle time between them; accordingly, M_1 processes all sublots in the interval $[0, np_1 + bs_1]$.
3. M_2 processes all sublots without any unnecessary delay and without interruption.

Let Σ be the set of all permutation schedules that possess these three properties. For each $\sigma \in \Sigma$, the completion time of the ith sublot on machine M_1, $C_{1i}(\sigma)$, can be written as

$$C_{1i}(\sigma) = is_1 + p_1 \sum_{j=1}^{i} x_j. \tag{1}$$

In contrast to machine M_1, there can be idle time on machine M_2 between consecutive sublots; this happens if sublot i has not finished yet on M_1, while sublot $i-1$ has

finished on M_2. Hence, the completion time of the ith sublot on machine M_2, $C_{2i}(\sigma)$, can be written as

$$C_{2i}(\sigma) = \max\{C_{1i}(\sigma), C_{2,i-1}(\sigma)\} + s_2 + p_2 x_i. \tag{2}$$

Note that for each $\sigma \in \Sigma$ and for each i $(i = 1, \ldots, b)$, we can identify a sublot q $(1 \leq q \leq i)$ such that

$$C_{2i}(\sigma) = q s_1 + p_1 \sum_{j=1}^{q} x_j + (i - q + 1) s_2 + p_2 \sum_{j=q}^{i} x_j.$$

We refer to the *last* sublot q that defines $C_{2i}(\sigma)$ in this way as the *pivotal* sublot. The pivotal sublot plays a central role in the exact dynamic programming algorithm we developed to find an optimal solution to our problem.

3 Dynamic programming algorithms

In this section, we introduce an exact dynamic programming algorithm based on the concept of pivotal sublot (Sect. 3.1). However, since the application of the algorithm might be limited to small instances, we also derived an approximate dynamic program (Sect. 3.2) and a local search procedure (Sect. 3.3) that run with lower time complexity and can be applied to larger instances.

3.1 An exact solution approach

To solve the problem introduced in Sect. 2 to optimality, we develop a forward implicit enumeration scheme that builds a complete schedule by adding sublots to the end of the current partial schedule one by one.

Let $\Pi(\alpha, \beta, q, j, k)$ be the set of all permutation schedules that possess the following properties:

1. there are exactly α items scheduled in β sublots;
2. $x_i \geq 1$ for $i = 1, \ldots, \beta$;
3. M_1 processes all β sublots in the interval $[0, \alpha p_1 + \beta s_1]$.
4. q is the pivotal sublot and $x_q = j$;
5. M_2 processes all sublots and items without unnecessary delay and without interruption;
6. there are k items scheduled in the first q sublots, that is, $\sum_{i=1}^{q} x_i = k$.

Since $x_i \geq 1$ for each $i = 1, \ldots, \beta$, the relevant ranges to consider for the values of α, β, q, j and k are: $\alpha = 1, \ldots, n$; $\beta = 1, \ldots, \alpha$; $q = 1, \ldots, \beta$; $j = 1, \ldots, \alpha - \beta + 1$; $k = j + q - 1, \ldots, \alpha - \beta + q$.

Let π^* be an optimal schedule for the original problem. We then have that

$$\pi^* \in \Pi(n, \beta, q, j, k)$$

for some β ($1 \leq \beta \leq n$), q ($1 \leq q \leq \beta$), j ($1 \leq j \leq n - \beta + 1$), and k ($j + q - 1 \leq k \leq n - \beta + q$).

Consider now any schedule $\pi \in \Pi(\alpha, \beta, q, j, k)$ for some α, β, q, j and k. We define such a schedule to be in state (α, β, q, j, k). Any schedule in this state must complete at time

$$t_1(\alpha, \beta, q, j, k) = \alpha p_1 + \beta s_1$$

on machine M_1, and complete on M_2 at time

$$t_2(\alpha, \beta, q, j, k) = kp_1 + qs_1 + (\alpha + j - k)p_2 + (\beta - q + 1)s_2.$$

To schedule the remaining $n - \alpha$ items, we have to consider only schedules in $\Pi(\alpha, \beta, q, j, k)$ that have minimal cost. Let $\pi(\alpha, \beta, q, j, k)$ be such a schedule and let $F(\alpha, \beta, q, j, k)$ be its corresponding cost. Notice that in our dynamic program we set $F(\alpha, \beta, q, j, k) = +\infty$ for all the combinations of α, β, q, j, k that do not admit a feasible schedule. These cases refer also to situations where at least one of the values of α, β, q, j, k is zero, except for the state (0,0,0,0,0) for which we consider a dummy schedule with zero cost and zero completion time.

Schedule $\pi(\alpha, \beta, q, j, k)$ must have been obtained by appending a sublot with i items ($1 \leq i \leq \alpha$) to some previous schedule $\pi(\alpha - i, \beta - 1, x, y, z)$ with cost $F(\alpha - i, \beta - 1, x, y, z)$ for specific values of x ($1 \leq x \leq \beta - 1$), y ($1 \leq y \leq \alpha - i - \beta + 2$), and z ($y + x - 1 \leq z \leq \alpha - i - \beta + 1 + x$).

In fact, we can differentiate two cases:

1. $q < \beta$. Effectively, this means that the addition of the βth sublot with i items does not change the pivotal sublot. The previous schedule must then be of the type $\pi(\alpha - i, \beta - 1, q, j, k)$, and this can be the case if and only if $t_1(\alpha, \beta, q, j, k) \leq t_2(\alpha - i, \beta - 1, q, j, k)$. Let $I(\alpha, \beta, q, j, k)$ be the set of indices i ($1 \leq i \leq \alpha$) for which this condition is met. Then, if $I(\alpha, \beta, q, j, k)$ is not empty, we have that

$$F(\alpha, \beta, q, j, k) = \min_{i \in I(\alpha, \beta, q, j, k)} \{F(\alpha - i, \beta - 1, q, j, k) + i \cdot t_2(\alpha, \beta, q, j, k)\}.$$

2. $q = \beta$. Hence, the pivotal sublot changes, the βth sublot becomes the pivotal sublot. This implies also that $j = i$ and $k = \alpha$ and the resulting schedule must be $\pi(\alpha, \beta, \beta, j, \alpha)$. The previous schedule must then be of the type $\pi(\alpha - j, \beta - 1, x, y, z)$ for some values of x, y, z. This can be the case if and only if $t_1(\alpha, \beta, \beta, j, \alpha) > t_2(\alpha - j, \beta - 1, x, y, z)$. Let $I_0(\alpha, \beta, j)$ be the set of all indices x, y and z for which this is the case. Then, if $I_0(\alpha, \beta, j)$ is not empty, we have that

$$F(\alpha, \beta, q, j, k) = \min_{x, y, z \in I_0(\alpha, \beta, j)} \{F(\alpha - j, \beta - 1, x, y, z) + j \cdot t_2(\alpha, \beta, \beta, j, \alpha)\}.$$

The dynamic programming algorithm is then as follows. The initialization is

$$F(\alpha, \beta, q, j, k) = \begin{cases} 0, & \text{if } \alpha = \beta = q = j = k = 0, \\ +\infty, & \text{otherwise.} \end{cases}$$

The recursion is then to calculate for $\alpha = 1, \ldots, n$, $\beta = 1, \ldots, \alpha$, $q = 1, \ldots, \beta$, $j = 1, \ldots, \alpha - \beta + 1$, $k = j + q - 1, \ldots, \alpha - \beta + q$:

$$F(\alpha, \beta, q, j, k) = \begin{cases} \min_{i \in I}\{F(\alpha - i, \beta - 1, q, j, k) + it_2(\alpha, \beta, q, j, k)\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } q < \beta; \\ \min_{x,y,z \in I_0}\{F(\alpha - j, \beta - 1, x, y, z) + jt_2(\alpha, \beta, \beta, j, \alpha)\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } q = \beta, k = \alpha. \end{cases}$$

where no update is performed if the minimum is taken over an empty set. The optimal solution value is given by the minimum value of $F(n, \beta, q, j, k)$. The corresponding schedule can be derived by backtracking the updates of the values of $F(\alpha, \beta, q, j, k)$ during the execution of the algorithm. This dynamic programming algorithm can be implemented to run in $O(n^6)$ time and $O(n^5)$ space. In fact, the size of the ranges of α, β, q, j, k is in the order of $O(n)$. The recursion executes five nested for-loops over α, β, q, j, k and, for any given assignment of α, β, q, j, k, computes the minimum over set I in $O(n)$. Also, the recursion computes the minimum over set I_0 in $O(n^3)$ and this comparison is performed within three nested for-loops over α, β, j (as $q = \beta$ and $k = \alpha$ in that case). Thus, the overall time complexity is $O(n^6)$. The largest memorization requirements are related to the storage of the arrays $F(\alpha, \beta, q, j, k)$, thus giving a space complexity of $O(n^5)$.

However, notice that the proposed algorithm cannot be considered a polynomial time algorithm. Each problem instance can be, in fact, defined by entries s_1, s_2, p_1, p_2 related to the two machines and the number of items n. Since the length of the encoded input consists of five elements only, a polynomial-time algorithm for the problem, if any exists, cannot have a running time which is a function of n, even if it is polynomial.

3.2 An approximate dynamic programming algorithm

Time and space complexities of the optimzation algorithm limit its application to large instances of the problem. Specifically, preliminary tests showed that the algorithm manages to solve instances with up 50 items with a computation time of about 1000 s.

In the light of these considerations, we propose a less computationally demanding dynamic program that does not guarantee to find an optimal solution but runs with a much lower running time of $O(n^3)$. As shown in the computational result section, this algorithm turns out to be very competitive to the approaches in the literature while requiring a very limited computational effort.

The idea is to compute solutions with b sublots and up to n items by analyzing the schedules obtained in sub-problems with $(b - 1)$ sublots and a given subset of items. Similarly as before, we denote by $F(\alpha, \beta)$ the cost of a solution where α items are scheduled in β sublots and, with a slight abuse of notation, by $C_2(\alpha, \beta)$ the associated completion time.

In an initialization step, we first compute $F(\alpha, 1)$ and $C_2(\alpha, 1)$ ($\alpha = 1, \ldots, n$), namely we consider the case where items are packed in one sublot.

1. For $\alpha = 1, \ldots n$:

$$C_2(\alpha, 1) = s_1 + \alpha p_1 + s_2 + \alpha p_2;$$
$$F(\alpha, 1) = \alpha C_2(\alpha, 1).$$

After this initialization step, the following recursion is iteratively applied within the following three nested for-loops (with B being an auxiliary parameter):

2. For $\beta = 2, \ldots, n$ then
   for $\alpha = \beta, \ldots, n$, set $B = \infty$ and for $i = 1, \ldots, (\alpha - \beta + 1)$:

$$C_2' = \begin{cases} \beta s_1 + \alpha p_1 + s_2 + i p_2 & \text{if } \beta s_1 + \alpha p_1 \geq C_2(\alpha - i, \beta - 1) \\ C_2(\alpha - i, \beta - 1) + s_2 + i p_2 & \text{otherwise.} \end{cases}$$

   If $i C_2' + F(\alpha - i, \beta - 1) < B$, then

$$C_2(\alpha, \beta) = C_2',$$
$$F(\alpha, \beta) = i C_2(\alpha, \beta) + F(\alpha - i, \beta - 1),$$
$$B = F(\alpha, \beta).$$

The two outer loops consider solutions where α items are scheduled in β sublots (with $\alpha \geq \beta$ as each sublot makes sense if processes at least one item). To this extent, in the inner loop, the insertion of i items in the last sublot β is evaluated by considering subproblems $C_2(\alpha - i, \beta - 1)$. Quantity C_2' denotes the corresponding completion time and the last *if* condition (i.e., if $i C_2' + F(\alpha - i, \beta - 1) < B$) simply keeps track of the best solution found along the iterations.

The solution value returned by the algorithm is the minimum value $F(n, \beta)$ for $\beta = 1, \ldots, n$ and it can be computed during the execution of the algorithm. The corresponding schedule can be recovered by a backtracking strategy. Clearly, the dynamic program optimizes only "locally" by selecting the best option for a given subproblem on the basis of previous greedy choices. Therefore, the algorithm cannot guarantee that a globally optimal solution will be obtained.

This algorithm has a time complexity of $O(n^3)$ and a space complexity of $O(n^2)$: the three for-loops require $O(n^3)$ elementary operations to be performed, giving the time complexity; the size of both arrays $C_2(\alpha, \beta)$ and $F(\alpha, \beta)$ is $O(n^2)$.

3.3 A local search procedure

In this section, we propose a dynasearch-like local search procedure to possibly improve a given feasible solution $\sigma_b = (x_1, \ldots, x_b)$.

Let C_{1i}' and C_{2i}' ($i = 1, \ldots, b$) be, respectively, the sublot completion times on machines M_1 and M_2 of such a solution. This solution may be improved by local search, for instance by simply moving one or more items from one sublot to another.

The general principle of dynasearch is to find the best solution in a local search neighborhood of exponential size by use of a dynamic programming algorithm by allowing only so-called independent moves (Congram et al. 2002). For our lot streaming problem, moving items from sublot i to sublot k and from sublot f to sublot l constitute two *independent* moves if $\max\{i,k\} < \min\{f,l\}$ or $\min\{i,k\} > \max\{f,l\}$. For our problem, however, it is computationally intractable to search the dynasearch neighborhood, and we therefore propose to reduce this neighborhood by excluding the solutions in which the completion time of sublot k on machine M_2 is later than $C'_{2,k}$.

This intervention allows us to use a dynamic programming algorithm to find the best solution in this restricted local search neighborhood as follows. Let $\coprod(k)$ $(k \le b)$ be the set containing all schedules with k sublots in which the completion time of sublot k on machine M_2 is smaller than $C'_{2,k}$. Therefore, each schedule in $\coprod(k)$ is at least as good as the initial partial schedule $\sigma_k = (x_1, \ldots, x_k)$ both in terms of total flow time and makespan. Indeed, a smaller completion time on M_2 for sublot k does not necessarily mean a smaller total flow time. However, since this condition is valid for any $k \le b$, each sublot has a completion time on M_2 smaller that the initial schedule, thus implying a better total flow time. Any such schedule in $\coprod(k)$ is said to be in state k and completes at time $t_1(k)$ and $t_2(k)$ on machine M_1 and machine M_2, respectively.

Let $\beta(k)$ be the best schedule in state k and $F(k)$ its corresponding cost. Schedule $\beta(k)$ must have been obtained by appending $(k-i)$ sublots to some previous schedule $\beta(i)$ with value $F(i)$. This can be done in two different ways:

1. If $i = k-1$, sublot k (with size $x_k \in \sigma_b$) is added to $\beta(k-1)$. In this case,

$$t_1(k) = t_1(k-1) + x_k p_1 + s_1,$$
$$t_2(k) = \max\{t_1(k-1) + x_k p_1 + s_1, t_2(k-1)\} + x_k p_2 + s_2.$$

2. If $i \le k-2$, we can try to interchange items between the first and the last sublot in the partial schedule (i.e., between sublot $(i+1)$ and sublot k). Let $C^{i,j}_{1,h}$ and $C^{i,j}_{2,h}$ denote, in state k, the completion time of sublot h on machine M_1 and M_2, respectively, if j items are moved from sublot $i+1$ to sublot k. Then we have

$$F(k) = F(i) + (x_{i+1} - j)C^{i,j}_{2,i+1} + x_{i+2}C^{i,j}_{2,i+2} + \cdots + (x_k + j)C^{i,j}_{2,k}.$$

The dynamic programming algorithm is as follows. The initialization is

$$F(0) = 0, \quad t_1(0) = 0, \quad t_2(0) = 0,$$
$$F(1) = x_1(x_1 p_1 + s_1 + x_1 p_2 + s_2), \quad t_1(1) = x_1 p_1 + s_1,$$
$$t_2(1) = x_1 p_1 + s_1 + x_1 p_2 + s_2.$$

The recursion for $k = 2, \ldots, n$ is

$$F(k) = \min_{0 \le i \le k-1} \begin{cases} F(i) + x_k(\max\{t_1(k-1) + x_k p_1 + s_1, t_2(k-1)\} + x_k p_2 + s_2), & i = k-1, \\ \min_{-x_k < j < x_{i+1}}\{F(i) + (x_{i+1} - j)C^{i,j}_{2,i+1} + \\ \qquad x_{i+2}C^{i,j}_{2,i+2} + \cdots + (x_k + j)C^{i,j}_{2,k}\}, & 0 \le i \le k-2, \end{cases}$$

$$C_{1,h}^{i,j} = \begin{cases} t_1(i) + (x_{i+1} - j)p_1 + s_1, & h = i+1, \\ C_{1,h-1}^{i,j} + x_h p_1 + s_1, & i+2 \le h \le k-1, \\ C_{1,k-1}^{i,j} + (x_k + j)p_1 + s_1, & h = k, \end{cases}$$

$$C_{2,h}^{i,j} = \begin{cases} \max\{t_1(i) + (x_{i+1} - j)p_1 + s_1, t_2(i)\} + (x_{i+1} - j)p_2 + s_2, & h = i+1, \\ \max\{C_{1,h-1}^{i,j} + x_h p_1 + s_1, C_{2,h-1}^{i,j}\} + x_h p_2 + s_2, & i+2 \le h \le k-1, \\ \max\{C_{1,k-1}^{i,j} + (x_k + j)p_1 + s_1, C_{2,k-1}^{i,j}\} + (x_k + j)p_2 + s_2, & h = k. \end{cases}$$

To compute $t_1(k)$ and $t_2(k)$, let i and j be the values for which function $F(k)$ is minimum, i.e.,

$$(i, j) = \arg\min F(k).$$

In the case $i = k - 1$, the value of j is immaterial. Hence,

$$t_1(k) = \begin{cases} t_1(k-1) + x_k p_1 + s_1, & k = i+1, \\ C_{1,k}^{i,j}, & 2 \le k \le i+2, \end{cases}$$

$$t_2(k) = \begin{cases} \max\{t_1(k-1) + x_k p_1 + s_1, t_2(k-1)\} + x_k p_2 + s_2, & k = i+1, \\ C_{2,k}^{i,j}, & 2 \le k \le i+2. \end{cases}$$

The final solution value is then equal to $F(n)$, and the corresponding solution can be found by backtracking. The dynasearch algorithm has a computational complexity of $O(n^3)$ per iteration.

4 Lagrangian lower bounds

In addition to the algorithms presented in Sect. 3, we propose lower bounds for the problem, based on Lagrangian relaxation. These Lagrangian lower bounds turn out to be very effective, allowing us to verify the optimality of the solutions given by the proposed approximation algorithm for many benchmark instances, as it will be discussed in Sect. 5.

The main idea behind Lagrangian relaxation is to see an NP-hard combinatorial optimization problem as an "easy-to-solve" problem complicated by a number of "nasty" side constraints. *Dualizing* these nasty constraints, that is, putting them into the objective function, each weighted by a given Lagrangian multiplier, gives a Lagrangian relaxation problem, and its optimal solution gives a lower bound on the optimal solution value for any given vector of multipliers (Held and Karp 1970, 1971). Note that this Lagrangian solution may not be feasible, let alone optimal, for the original problem.

Lagrangian relaxation has been successfully applied to a wide range of NP-hard combinatorial optimization problems (Geoffrion 1974; Shapiro 1979; Fisher 1981), but, as far as we know, it has never been applied to a lot streaming problem.

Using the structural properties of an optimal schedule and the notation derived in Sect. 2, we let each term x_i denote the size of sublot i ($i = 1, \ldots, b$), and let C_{1i} and C_{2i} denote the completion time of sublot i ($i = 1, \ldots, b$) on machine M_1 and machine M_2, respectively. The lot streaming problem of minimizing total flow time in

a two-machine flow shop with consistent sublots sizes and attached set up times can then be formulated as the following integer quadratic programming problem:

$$\min \sum_{i=1}^{b} x_i C_{2i}$$

s.t.

$$C_{1i} = \sum_{j=1}^{i} p_1 x_j + i s_1 \qquad \text{for } i = 1, \ldots, b \tag{3}$$

$$C_{2i} \geq C_{1i} + p_2 x_i + s_2 \qquad \text{for } i = 1, \ldots, b \tag{4}$$

$$C_{2i} \geq C_{2,i-1} + p_2 x_i + s_2 \qquad \text{for } i = 2, \ldots, b \tag{5}$$

$$\sum_{i=1}^{b} x_i = n \tag{6}$$

$$x_i \in Z^+, \; C_{1i}, C_{2i} \in R^+ \qquad \text{for } i = 1, \ldots, b. \tag{7}$$

The objective is to minimize the sum of the item completion times. Constraints (3) ensure that machine M_1 processes all sublots consecutively in the interval $[0, np_1 + bs_1]$, without any idle time in between. Constraints (4) force each sublot i not to start on M_2 before its completion on M_1. Constraints (5) forbid machine M_2 from processing more than one sublot at a time; in other words, they are machine M_2 capacity constraints. Constraint (6) represents the fact that all n items have to be allocated to sublots. Constraints (7) force the sublot sizes to be positive integral variables because each sublot must include at least one item and state that the completion times are non-negative variables.

Analyzing the structure of the above model, it is apparent that the problem would be relatively fast to solve if b, the number of sublots, would be known a-priori and if conditions (4) or (5) were absent. Indeed, if b were given and if the precedence constraints (4) were absent, then the problem would decompose into two independent single machine batching problems with the requirement of having consistent sublot sizes the only constraint linking them together. This problem can be solved using a straightforward dynamic programming algorithm (Brucker et al. 1998). Alternatively, if b were given and if the capacity constraints (5) for machine M_2 were absent, then conditions (4) would hold with equality in any optimal solution, and the problem would effectively decompose into a single machine batching problem solvable by a similar dynamic programming algorithm.

Hence, in the following, we work with fixed b and derive and analyze two Lagrangian relaxation problems, one obtained by relaxing the precedence constraints (4), the other obtained by relaxing the machine M_2 capacity constraints (5). Then we would need to solve the two Lagrangian problems for every possible value of b, i.e., for $b = 1, \ldots, n$, to find the true lower bound on the optimal solution value for the original lot streaming problem.

4.1 Lagrangian relaxation of the precedence constraints

In this section, we investigate the Lagrangian relaxation obtained by dualizing the precedence constraints (4) with a vector of Lagrangian multipliers $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_b) \geq 0$. The Lagrangian relaxation problem $L^1_{(\boldsymbol{\lambda},b)}$, for b sublots, is then the problem of finding $L^1(\boldsymbol{\lambda}, b)$, which is the optimal solution of

$$\min \sum_{i=1}^{b} (x_i - \lambda_i) C_{2i} + \sum_{i=1}^{b} \lambda_i (C_{1i} + p_2 x_i + s_2) \tag{8}$$

$$\text{s.t. } \lambda_i \leq x_i \qquad \text{for } i = 1, \ldots, b \tag{9}$$

$$x_i \leq u_i \qquad \text{for } i = 1, \ldots, b \tag{10}$$

$$(3), (5), (6), (7),$$

where constraints (9) are used to avoid the trivial lower bound $L^1(\boldsymbol{\lambda}, b) = -\infty$ achieved by taking $\lambda_i > x_i$ for some i. Instead, constraints (10) are used to strengthen the Lagrangian lower bounds thus improving the computational efficiency of the solution. Upper bounds u_i on the number of items in sublot i are derived in "Appendix A".

From Lagrangian theory, it is well known that $L^1(\boldsymbol{\lambda}, b)$ is a lower bound on optimal solution value of the original lot streaming problem with a fixed number of sublots b, for any given vector $\boldsymbol{\lambda} \geq 0$. However, we consider here the further restriction $\lambda_i \leq 1$ for $i = 1, \ldots, b$ to efficiently solve the Lagrangian problem by dynamic programming. This aspect is discussed at the end of the section.

As claimed before, the Lagrangian problem $L^1_{(\boldsymbol{\lambda},b)}$ can be solved by a dynamic programming algorithm, similar to the one used by Brucker et al. (1998) for the single machine batching problem to minimize total completion time, which uses a forward implicit enumeration scheme that successively adds sublots to the end of the current (partial) schedule. Let $\Sigma(j, i)$ $(i \leq b)$ be the set of all permutation schedules that are feasible for problem $L^1(\boldsymbol{\lambda}, b)$ with j items scheduled in i sublots. We define any such schedule to be in state (j, i). A schedule in state (j, i) must complete at time $t_1(j, i) = is_1 + jp_1$ on machine M_1 and at time $t_2(j, i) = s_1 + p_1 + is_2 + jp_2$ on machine M_2 (remember that M_2 cannot start processing the first sublot before time $s_1 + p_1$).

To schedule the remaining $(n - j)$ items, we have to consider only schedules in $\Sigma(j, i)$ that have minimal cost. Let $\sigma(j, i)$ be such a schedule and let $F_i(j)$ be its corresponding cost. Schedule $\sigma(j, i)$ must have been obtained by appending a sublot with $(j - k)$ items to some previous schedule $\sigma(k, i - 1)$ with cost $F_{i-1}(k)$. In this case, we must have that

$$F_i(j) = F_{i-1}(k) + (j - k - \lambda_i) t_2(j, i) + \lambda_i (t_1(j, i) + p_2(j - k) + s_2).$$

The dynamic programming recursion is then as follows. The initialization is

$$F_i(j) = \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

These values are updated recursively, to

$$F_i(j) = \min_{i-1 \le k \le j-1} \{F_{i-1}(k) + (j - k - \lambda_i)t_2(j,i) + \lambda_i(t_1(j,i) + p_2(j-k) + s_2)\},$$

for $j = 1, \ldots, n$ and $i = 1, \ldots, b$, taken in lexicographic order. Notice that for the correctness of the recursion to compute lower bounds, each term $(j - k - \lambda_i)$ cannot be negative due to the non-negativity requirement of terms $(x_i - \lambda_i)$ in the Lagrangian model. Hence, as the maximum value of k is $j - 1$, we must consider only Lagrangian multipliers not larger than 1, i.e., $\lambda_i \le 1$ for $i = 1, \ldots, b$. The optimal solution value, for a given b, is $F_b(n)$, i.e., $L^1(\boldsymbol{\lambda}, b) = F_b(n)$, and the optimal solution can be found by backtracking. The dynamic programming algorithm can be implemented to run in $O(bn^2)$ time.

$L^1(\boldsymbol{\lambda}, b)$ is a lower bound for any $0 \le \lambda_i \le 1$, but we are interested in finding the best Lagrangian multipliers $0 \le \lambda_i^*(b) \le 1$ that give the best lower bound for fixed b, i.e., $L^1(\boldsymbol{\lambda}^*(b), b) = \max_{0 \le \lambda \le 1} L^1(\boldsymbol{\lambda}, b)$. To approximate these optimal values, we will use the well known subgradient method (Held and Karp 1971; Held et al. 1974; Fisher 1985). The overall best Lagrangian lower bound is given by $\min_{1 \le b \le n} L^1(\boldsymbol{\lambda}^*(b), b)$.

4.2 Lagrangian relaxation of the capacity constraints

An alternative Lagrangian relaxation can be obtained by dualizing the capacity constraints (5) for machine M_2. As in the previous type of Lagrangian relaxation, we assume an a-priori fixed value for b. Relaxing constraints (5) allows sublots to overlap on M_2 and hence the precedence constraints (4) will hold with equality in any optimal solution for this Lagrangian problem; in other words, each sublot i $(i = 1, \ldots, b)$ will start its execution on machine M_2 as soon as it is finished on M_1 and the attached setup has been performed. To dualize constraints (5), we introduce a vector of Lagrangian multipliers $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{b+1}) \ge 0$ where multipliers $\mu_2, \ldots, \mu_b$ are associated with constraints (5) while dummy multipliers μ_1 and μ_{b+1} are introduced for notational convenience and are set to zero. This leads to the Lagrangian problem $L^2_{(\boldsymbol{\mu},b)}$ of finding

$$L^2(\boldsymbol{\mu}, b) = \min \sum_{i=1}^{b} x_i C_{2i} + \sum_{i=2}^{b} \mu_i (C_{2,i-1} + x_i p_2 + s_2 - C_{2i}) \tag{11}$$

$$\text{s.t. } \mu_i - \mu_{i+1} \le x_i \qquad \text{for } i = 1, \ldots, b \tag{12}$$

$$(3), (4), (6), (7), (10), \text{ with } \mu_1 = 0, \mu_{b+1} = 0,$$

where constraints (12) are used to allow only those values of $\boldsymbol{\mu}$ for which the Lagrangian objective function is non-decreasing in the sublot completion times C_{2i} (thus avoiding that the objective function goes to $-\infty$). In fact, re-arranging terms in

the objective function, $L^2(\boldsymbol{\mu}, b)$ can be re-written as

$$L^2(\boldsymbol{\mu}, b) = \min \sum_{i=1}^{b} (x_i - \mu_i + \mu_{i+1}) C_{2i} + \sum_{i=2}^{b} \mu_i (x_i p_2 + s_2)$$
$$\text{s.t. } \mu_i - \mu_{i+1} \leq x_i \qquad \text{for } i = 1, \ldots, b$$
$$(3), (4), (6), (7), (10), \text{ with } \mu_1 = 0,\ \mu_{b+1} = 0. \tag{13}$$

The Lagrangian problem $L^2_{(\boldsymbol{\mu},b)}$ can be solved by a dynamic programming algorithm similar to the one used for solving problem $L^1_{(\boldsymbol{\lambda},b)}$. Let $\Pi(j, i)$ $(i \leq b)$ be the set of all permutation schedules that are feasible for $L^2(\boldsymbol{\mu}, b)$ with j items scheduled in i sublots. We define any such schedule to be in state (j, i). All schedules in state (j, i) must complete at time $v_1(j, i) = is_1 + jp_1$ on machine M_1 and, since M_2 can process multiple sublots at the same time (since constraints (4) hold with equality in any optimal solution of the Lagrangian problem), all such schedules must complete at time $v_2(j, i) = v_1(j, i) + s_2 + lp_2$ on machine M_2, with l the number of items in the ith sublot.

To schedule the remaining $(n - j)$ items, we have to consider only schedules in $\Pi(j, i)$ that have minimal cost. Let $\pi(j, i)$ be such a schedule and let $G_i(j)$ be its corresponding cost. Schedule $\pi(j, i)$ must have been obtained by appending a sublot with $(j - k)$ items to some previous schedule $\pi(k, i - 1)$ with cost $G_{i-1}(k)$. In this case, we must have that

$$G_i(j) = G_{i-1}(k) + (j - k - \mu_i + \mu_{i+1}) v_2(j, i) + \mu_i ((j - k) p_2 + s_2).$$

The dynamic programming recursion is as follows. The initialization is

$$G_i(j) = \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

These values are updated recursively, to

$$G_i(j) = \min_{i-1 \leq k \leq j-1} \{G_{i-1}(k) + (j - k - \mu_i + \mu_{i+1}) v_2(j, i) + \mu_i ((j - k) p_2 + s_2)\},$$

for $j = 1, \ldots, n$ and $i = 1, \ldots, b$, taken in lexicographic order. The optimal solution value is then $G_b(n)$, and the optimal solution can be found by backtracking. The dynamic programming algorithm can be implemented to run in $O(bn^2)$ time. As discussed in the Lagrangian relaxation of the precedence constraints, the correctness of the recursion to compute lower bounds is guaranteed if each term $(j - k - \mu_i + \mu_{i+1})$ is non-negative. In this case, as the maximum value of k is $j - 1$, we must have $\mu_i - \mu_{i+1} \leq 1$ for $i = 1, \ldots, b$. Starting from the last sublot b, we have $\mu_{b+1} = 0 \implies \mu_b \leq 1$. For sublot $b - 1$, we have $\mu_{b-1} \leq 1 + \mu_b \leq 2$. Applying the same reasoning for sublots $b - 2, \ldots, 1$ implies that $L^2(\boldsymbol{\mu}, b)$ is a lower bound for any μ that satisfies $0 \leq \mu_\ell \leq b - \ell + 1$ $(\ell = 1, \ldots, b)$. Exactly as in the case of $L^1(\boldsymbol{\lambda}, b)$, we use subgradient methods to approximate the best lower bound for fixed b, i.e., to

find the vector $\boldsymbol{\mu}^*(b)$ and hence $L^2(\boldsymbol{\mu}^*(b), b) = \max_{1\le\mu\le b} L^2(\boldsymbol{\mu}, b)$. The best overall Lagrangian lower bound will be given by $\min_{1\le b\le n} L^2(\boldsymbol{\mu}^*(b), b)$.

4.3 Computation of the lower bounds

Clearly, the best lower bound LB on the optimal total completion time is given by the maximum of the two Lagrangian relaxations, i.e.,

$$LB = \max\{L^1(\boldsymbol{\lambda}^*), L^2(\boldsymbol{\mu}^*)\},$$

where

$$L^1(\boldsymbol{\lambda}^*) = \min_{1\le b\le n} \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b),$$
$$L^2(\boldsymbol{\mu}^*) = \min_{1\le b\le n} \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b).$$

Notice that from the structure of both Lagrangian relaxations, it follows that

$$\min_b L^1(\mathbf{0}, b) \le \min_b \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b) \le \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b'),$$
$$\min_b L^2(\mathbf{0}, b) \le \min_b \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b) \le \max_{\boldsymbol{\mu}} L^1(\boldsymbol{\mu}, b''),$$

for any b' and b''. Hence, we have that

$$L^1(\boldsymbol{\lambda}^*) \in \left[L^1(\mathbf{0}, b'_0),\ \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b'_0) \right], \quad \text{with } b'_0 = \{b | \min_b L^1(\mathbf{0}, b)\},$$

and

$$L^2(\boldsymbol{\mu}^*) \in \left[L^2(\mathbf{0}, b''_0),\ \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b''_0) \right], \quad \text{with } b''_0 = \{b | \min_b L^2(\mathbf{0}, b)\}.$$

This implies that if $\min_b L^1(\mathbf{0}, b) > \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b''_0)$, we need to solve $\min_{1\le b\le n} \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b)$ only. By the same token, if $\min_b L^2(\mathbf{0}, b) > \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b'_0)$, we need to solve $\min_{1\le b\le n} \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b)$ only. We outline the procedure to compute the best bound from Lagrangian relaxations in the following two steps:

- *Step 1*. Solve $\min_b L^1(\mathbf{0}, b)$, $\min_b L^2(\mathbf{0}, b)$, $\max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b'_0)$ and $\max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b''_0)$.
- *Step 2*. If $\min_b L^1(\mathbf{0}, b) > \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b''_0)$, solve $\min_{1\le b\le n} \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b)$. Else if $\min_b L^2(\mathbf{0}, b) > \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b'_0)$, solve $\min_{1\le b\le n} \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b)$.
  Else, solve $\max\{\min_b \max_{\boldsymbol{\lambda}} L^1(\boldsymbol{\lambda}, b), \min_b \max_{\boldsymbol{\mu}} L^2(\boldsymbol{\mu}, b)\}$. In each iteration, use subgradient method to find the best Lagrangian multiplier $\boldsymbol{\lambda}^*$ and/or $\boldsymbol{\mu}^*$.

Finally notice that the range of the number of sublots $1 \le b \le n$ can be further narrowed by the procedures presented in "Appendix A".

5 Computational results

We tested our heuristic dynamic programming algorithm coupled with one step of the local search procedure on all the 40 instances described in Bukchin et al. (2002).

Tables 1 and 2 list our and Bukchin et al. (2002) computational results for each of the 40 instances, all of which had $n = 500$. Our tests were performed on an Intel i7 CPU @ 2.4 GHz with 8 GB of RAM. The code was implemented in the C++ programming language. In the tested instances, the heuristic algorithm computed all the solutions in about 3 s, requiring at most 0.08 s in the worst case and an average time of 0.07 s, while the total computation time for devising the Lagrangian lower bounds was about 6.5 minutes for all the 40 instances (with an average time of about 10 s and a maximum time of about 13 s).

The tables show, for each instance, our lower bound (LB_O) and our upper bound (UB_O), that is, the solution found by the heuristic algorithm coupled with one step of the local search procedure. They also show the results obtained by Bukchin et al. (2002), that is, their lower bound (LB_B) and the solution value of their best solution for the continuous version of each instance (UB_B).

Since Bukchin et al. (2002) addressed the continuous version of the problem (assuming that the fractional solution could be rounded so as to obtain a good feasible solution to the discrete version of the problem), the values UB_O and the UB_B are not directly comparable. However, since the upper bounds UB_O and UB_B are computed by finding a feasible discrete and continuous solution, respectively, and the discrete solution is still a feasible solution for the continuous case, as expected, in general, $UB_B \leq UB_O$.

Table 1 Instances with $LB_O = UB_O$

Problem	s_1	s_2	p_1	p_2	Our results		Buckhin's		
					LB_O	UB_O	LB_B	UB_B	UB_R
1	6.72	2.32	0.77	0.69	278.3**	278.3	198.4	278.3	278.3
2	9.01	3.4	0.97	0.46	337.5**	337.5	252.0	337.5	337.5
4	7.70	3.58	0.98	0.74	344.2**	344.2	253.8	344.2	344.2
5	1.70	1.03	0.61	0.22	182.6**	182.6	154.6	182.6	182.6
7	7.45	7.69	0.58	0.15	210.5**	210.5	153.7	210.5	210.5
8	1.20	6.04	0.77	0.09	221.7**	221.7	194.9	221.7	221.7
9	6.11	7.44	0.68	0.34	241.3**	241.3	176.7	241.3	241.3
21	60.52	25.86	0.89	0.42	495.3**	495.3	282.3	495.3	495.3
22	93.52	56.48	0.43	0.03	356.6**	356.6	200.5	356.6	356.6
23	97.84	16.77	0.81	0.61	567.1**	567.1	299.5	567.1	567.1
24	77.79	61.93	0.66	0.02	423.6**	423.6	243.1	423.6	423.6
25	78.05	23.18	0.61	0.56	461.1**	461.1	230.1	461.1	461.1
29	93.19	98.98	0.4	0.03	386.8**	386.8	192.9	386.8	386.8
36	88.40	5.77	0.38	0.56	389.5**	389.5	234.1	389.5	389.5
37	40.90	7.9	0.03	0.07	93.1**	93.1	65.9	93.1	93.1
40	53.62	2.81	0.54	0.76	386.9**	386.9	247.3	386.9	386.9

Moreover, the results indicate that the difference between UB_B and UB_O is not too large. For a fairer comparison of the results, we also rounded the Bukchin's solution UB_B to obtain a feasible solution for the discrete problem. The rounded solution is reported in column UB_R of both tables.

Our algorithm finds an optimal solution for 16 out of 40 instances. These instances, in which our lower and upper bound values are equal, are presented in Table 1. We observe that in all these cases the continuous upper bound UB_B achieved the same value. This curious result may indicate more inherent structure in the problem than current theory reveals, though the relationship $UB_O = UB_B$ does not conversely identify an optimal value, as we see in the results below.

The results for the 24 remaining instances, whose solutions are not necessarily optimal, are presented in Table 2. The proposed algorithm compares very favorably with the rounded Bukchin's solutions, providing upper bounds that are at least as good in about 40% of the instances (10 instances). In the other instances, our algorithm

Table 2 Instances with $LB_O < UB_O$

Problem	s_1	s_2	p_1	p_2	Our results		Buckhin's		
					LB_O	UB_O	LB_B	UB_B	UB_R
3	6.21	4.5	0.97	0.95	339.1**	339.2	247.9	339.1	339.3
6	1.35	7.32	0.8	0.23	235.5**	235.6	200.6	235.6	235.6
10	1.16	3.82	0.43	0.35	136.2**	139.2	109	139.1	139.2
11	6.22	7.61	0.1	0.79	260.8	266.3	211.2	265.8	265.8
12	6.61	8.54	0.13	0.63	219.8	226.8	171.5	225.7	225.7
13	1.42	9.22	0.72	0.84	281.2	301.2	221.4	299.5	299.7
14	5.9	8.6	0.45	0.95	312.4	325.4	252.4	323.5	323.7
15	1.78	8.03	0.14	0.66	221.7	228.7	174.4	227.4	227.5
16	6.61	3.73	0.07	0.7	218.2	221.3	186.1	220.9	220.9
17	9.00	1.84	0.64	0.97	282.4	296.8	253.6	296.5	296.7
18	3.26	1.82	0.43	0.95	270.7	276.1	243.3	274.9	275.1
19	4.02	2.27	0.05	0.91	263.2	264.9	233.3	264.8	264.8
20	6.42	5.75	0.32	0.56	190.4	201.1	151.4	200.2	200.3
26	3.75	19.37	0.85	0.54	291.5**	300.3	217.3	300.4	300.4
27	52.52	76.76	0.97	0.31	539.0**	539.3	295.4	539.2	539.3
28	55.92	74.82	0.75	0.62	513.5**	528.6	242.7	528.5	528.7
30	63.52	86.24	0.9	0.54	580.4**	583.7	289.6	583.7	583.8
31	32.88	52.96	0.62	0.9	459.0	503.8	311.3	503	503.1
32	3.23	55.69	0.13	0.87	407.0	421.8	276.1	421.6	421.6
33	59.66	60.68	0.59	0.7	469.3**	496.1	296.4	496	496.1
34	1.14	16.98	0.36	0.97	344.9	358.9	261.3	357.6	357.6
35	3.85	69.37	0.48	0.64	377.4	414.5	233.4	414	414.0
38	96.77	32.6	0.02	0.63	368.3	370.6	285.9	370.6	370.6
39	73.35	42.22	0.47	0.53	421.1**	421.2	248	421.2	421.2

Table 3 Average computation times (s)

n	UB_O	UB_B
500	0.07	172.05
600	0.10	220.60
700	0.20	294.76
1000	0.81	605.40

performs slightly worse than the competing approach but still gives upper bounds close to Bukchin's upper bounds. Moreover, our algorithm requires much shorter computation times, as can be seen from Table 3, where the computation times of our solution approach and Bukchin's approach are compared as n increases.

Observe also that the results for each of the 40 instances provide evidence of the compelling performance of our Lagrangian bounds. In Tables 1 and 2 , symbol $**$ in column LB_O indicate that the best bound is $L^2(\mu^*(b), b)$ while in the other instances the best bound is $L^1(\lambda^*(b), b)$. It can be noticed that in all the instances where the optimal solution is found (Table 1), the best lower bound is L^2. In general, L^2 is the best in the 63% of the cases (25 instances out of 40). However, since the percentage of instances in which the best bound came from L^1 is not negligible (37%) and these cases cannot be a-priori identified, we need to compute both L^1 and L^2 and choose the best one for each instance. The computational tests confirm the strength of the proposed lower bounds that allow a proof of optimality for many of the solutions provided.

6 Conclusions

The contribution of this paper is to provide new solution approaches for solving a lot streaming problem. We have presented an exact dynamic programming approach and a heuristic dynamic programming approach, which can be coupled with a local search procedure to improve the approximate solution, for the single-job discrete lot streaming problem of minimizing total flow time with attached setup times in a two-machine flow shop. Lagrangian algorithms to compute strong lower bounds were also proposed.

The computational performance of the proposed heuristic algorithm is very compelling; we are able to find a provably optimal solution for many instances (40%) and high quality solutions for the remaining instances within few seconds of computation. Also, the derived lower bounds turn out to be much stronger than those found by Bukchin et al. (2002) for the continuous variant of the problem. The proposed bounds can then be used to assess the quality of solutions provided by other heuristics that might be designed for the problem.

Together with the contribution brought by the dynamic programming algorithms, our work marks the first time that Lagrangian relaxation is used for a lot streaming problem, and we believe that our approach could be extended to other lot streaming

problems, such as for example the one with a general number of machines or with multiple jobs.

Acknowledgements We wish to thank the two anonymous referees and the associate editor for their suggestions and comments that allowed us to improve the manuscript. This work has been partially supported by "Ministero dell'Istruzione, dell'Università e della Ricerca" Award "TESUN-83486178370409 finanziamento dipartimenti di eccellenza CAP. 1694 TIT. 232 ART. 6".

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix A: Upper bound on the number of items in a sublot

In the following, we give a procedure to compute the upper bound for the number of items in each sublot in an optimal solution and hence the minimum number of required sublots in an optimal solution.

Let σ_b and σ'_{b+1} be two solutions for the lot streaming problem. Let $\sigma_b = (x_1, \ldots, x_b)$, $\sigma'_{b+1} = (x'_1, \ldots, x'_{b+1})$, and

$$x_i = \begin{cases} x'_i & \text{if } 1 \le i \le k-1, \\ x'_k + x'_{k+1} & \text{if } i = k, \\ x'_{i+1} & \text{if } k+1 \le i \le b. \end{cases}$$

The completion times of sublot i and sublot i' on machine M_1 and M_2 can be computed as follows:

$$\begin{aligned} C'_{1,i} &= C_{1i}, \quad i < k, \\ C'_{1k} &= C_{1k} - p_1 x'_{k+1}, \\ C'_{1,i+1} &= C_{1i} + s_1, \quad i > k, \\ C_{2,i} &= C'_{2,i}, \quad i < k, \\ C'_{2i} &= \max\{C'_{1i}, C'_{2,i-1}\} + s_2 + p_2 x'_i, \quad i \ge k. \end{aligned}$$

We first prove the following two Lemmas.

Lemma 1 $C'_{2,k+1} \le C_{2k} + \max\{s_1, s_2\}$.

Proof

$$\begin{aligned} C'_{2,k+1} &= \max\{C'_{1,k+1}, C'_{2k}\} + s_2 + p_2 x'_{k+1} \\ &= \max\{C_{1k} + s_1, \max\{C'_{1k}, C'_{2,k-1}\} + s_2 + p_2 x'_k\} + s_2 + p_2 x'_{k+1} \\ &= \max\{C_{1k} + s_1, \max\{C'_{1k}, C_{2,k-1}\} + s_2 + p_2 x'_k\} + s_2 + p_2 x'_{k+1} \\ &= \max\{C_{1k} + s_1 + s_2 + p_2 x'_{k+1}, C_{1k} - p_1 x_{k+1} \\ &\quad + 2s_2 + p_2 x_k, C_{2,k-1} + 2s_2 + p_2 x_k\} \end{aligned}$$

$$\leq \max\{C_{1k}, C_{2,k-1}\} + s_2 + p_2 x_k + \max\{s_1, s_2\}$$
$$= C_{2k} + \max\{s_1, s_2\}. \quad \square$$

$\square$

Lemma 2 *For any $i \geq k$, $C'_{2,i+2} \leq C_{2,i+1} + \max\{s_1, s_2\}$.*

Proof Assume $C'_{2,i+1} \leq C_{2i} + \max\{s_1, s_2\}$. In that case, the following relations hold:

$$C_{2,i+1} + \max\{s_1, s_2\} = \max\{C_{1,i+1} + \max\{s_1, s_2\}, C_{2i} + \max\{s_1, s_2\}\} + s_2 + p_2 x_{i+1}$$
$$\geq \max\{C'_{1,i+2}, C'_{2,i+1}\} + s_2 + p_2 x'_{i+2} = C'_{2,i+2}.$$

Hence, if $C'_{2,i+1} \leq C_{2i} + \max\{s_1, s_2\}$, then $C'_{2,i+2} \leq C_{2,i+1} + \max\{s_1, s_2\}$. The lemma now follows considering that $C'_{2,k+1} \leq C_{2k} + \max\{s_1, s_2\}$ (Lemma 1). $\square$

Consider the total completion time of the sublots from $k+2$ to $b+1$ in solution σ_{b+1}. Using the equality $x_i = x_{i+1}$ for $i \geq k+1$ and the result of Lemma 2, we can write

$$\sum_{i=k+2}^{b+1} x'_i C'_{2i} = \sum_{i=k+1}^{b} x_i C'_{2,i+1}$$
$$\leq \sum_{i=k+1}^{b} x_i (C_{2i} + \max\{s_1, s_2\}),$$

and hence,

$$\sum_{i=k+2}^{b+1} x'_i C'_{2i} - \sum_{i=k+1}^{b} x_i C_{2i} \leq \max\{s_1, s_2\} P_k^+,$$

where $P_k^+ = \sum_{i=k+1}^{b} x_i$.
Now suppose σ_b is an optimal solution. Then, we must have that

$$\sum_{i=k}^{b} x_i C_{2i} \leq \sum_{i=k}^{b+1} x'_i C'_{2i}$$
$$\sum_{i=k+1}^{b} x_i C_{2i} + x_k C_{2k} \leq \sum_{i=k+2}^{b+1} x'_i C'_{2i} + x'_k C'_{2k} + x'_{k+1} C'_{2,k+1}$$
$$x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \leq \sum_{i=k+2}^{b+1} x'_i C'_{2i} - \sum_{i=k+1}^{b} x_i C_{2i} \leq \max\{s_1, s_2\} P_k^+.$$

Since σ_b is optimal, the minimum value of $x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1}$ must be therefore smaller than $\max\{s_1, s_2\} P_k^+$.

Since the following equations hold

$$x_k C_{2k} = (x'_k + x'_{k+1})(\max\{C_{1k}, C_{2,k-1}\} + p_2 x'_k + p_2 x'_{k+1} + s_2),$$

$$x'_k C'_{2k} = x'_k(\max\{C'_{1k}, C'_{2,k-1}\} + p_2 x'_k + s_2),$$
$$x'_{k+1} C'_{2,k+1} = x'_{k+1}(\max\{C'_{1,k+1}, C'_{2k}\} + p_2 x'_{k+1} + s_2),$$

we can rewrite $x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1}$ as

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = (x'_k + x'_{k+1}) \max\{C_{1k}, C_{2,k-1}\} + 2p_2 x'_k x'_{k+1} \\ &\qquad - x'_k \max\{C'_{1k}, C'_{2,k-1}\} - x'_{k+1} \max\{C'_{1,k+1}, C'_{2k}\}. \end{aligned}$$

We can now distinguish six different cases:

1. $C_{2,k-1} < C'_{1k} < C_{1k}$, $C'_{1,k+1} < C'_{2k}$

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = x'_{k+1} C_{1k} + (2p_2 + p_1) x'_k x'_{k+1} - x'_{k+1}(C_{1k} - p_1 x'_{k+1} + p_2 x'_k + s_2) \\ &\quad = x'_{k+1}((p_2 + p_1) x'_k + p_1 x'_{k+1} - s_2)). \end{aligned}$$

2. $C_{2,k-1} < C'_{1k} < C_{1k}$, $C'_{1,k+1} \geq C'_{2k}$

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = (x'_k + x'_{k+1}) C_{1k} + 2p_2 x'_k x'_{k+1} - x'_k (C_{1k} - p_1 x'_{k+1}) - x'_{k+1} C'_{1,k+1} \\ &\quad = x'_{k+1} C_{1k} + (2p_2 + p_1) x'_k x'_{k+1} - x'_{k+1}(C_{1k} + s_1) \\ &\quad = x'_{k+1}((2p_2 + p_1) x'_k - s_1). \end{aligned}$$

3. $C'_{1k} < C_{2,k-1} < C_{1k}$, $C'_{1,k+1} < C'_{2k}$

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = (x'_k + x'_{k+1})(C_{1k} - C_{2,k-1}) + p_2 x'_k x'_{k+1} - x'_{k+1} s_2 \\ &\quad \geq x'_{k+1}(p_2 x'_k - s_2). \end{aligned}$$

4. $C'_{1k} < C_{2,k-1} < C_{1k}$, $C'_{1,k+1} \geq C'_{2k}$

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = (x'_k + x'_{k+1}) C_{1k} + 2p_2 x'_k x'_{k+1} - x'_k C_{2,k-1} - x'_{k+1}(C_{1k} + s_1) \\ &\quad = x'_k (C_{1k} - C_{2,k-1}) + 2p_2 x'_k x'_{k+1} - x'_{k+1} s_1 \\ &\quad \geq x'_{k+1}(2p_2 x'_k - s_1). \end{aligned}$$

5. $C'_{1k} < C_{1k} < C_{2,k-1}$, $C'_{1,k+1} < C'_{2k}$

$$\begin{aligned} &x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\ &\quad = (x'_k + x'_{k+1}) C_{2,k-1} + 2p_2 x'_k x'_{k+1} - x'_k C_{2,k-1} \end{aligned}$$

$$
\begin{aligned}
&- x'_{k+1}(C_{2,k-1} + p_2 x'_k + s_2) \\
&= x'_{k+1}(p_2 x'_k - s_2).
\end{aligned}
$$

6. $C'_{1k} < C_{1k} < C_{2,k-1}$, $C'_{1,k+1} \geq C'_{2k}$

$$
\begin{aligned}
&x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \\
&= (x'_k + x'_{k+1}) C_{2,k-1} + 2p_2 x'_k x'_{k+1} - x'_k C_{2,k-1} - x'_{k+1}(C_{1k} + s_1) \\
&= x'_{k+1}(C_{2,k-1} - C_{1k}) + 2p_2 x'_k x'_{k+1} - x'_{k+1} s_1 \\
&\geq x'_{k+1}(2p_2 x'_k - s_1).
\end{aligned}
$$

Putting all the six cases together, we have that

$$
x_k C_{2k} - x'_k C'_{2k} - x'_{k+1} C'_{2,k+1} \geq \min\{x'_{k+1}(p_2 x'_k - s_2), x'_{k+1}(2p_2 x'_k - s_1)\},
$$

and if

$$
\min\{x'_{k+1}(p_2 x'_k - s_2), x'_{k+1}(2p_2 x'_k - s_1)\} > \max\{s_1, s_2\} P_k^+,
$$

the solution σ_b cannot be optimal and hence sublot k should be split. Thus, we get a upper bound u_k on x_k by solving the following minimization problem:

$$
\begin{aligned}
\min\ & u_k = x'_k + x'_{k+1} && (14)\\
\text{s.t. } & x'_{k+1}(p_2 x'_k - s_2) > \max\{s_1, s_2\} P_k^+ && (15)\\
& x'_{k+1}(2p_2 x'_k - s_1) > \max\{s_1, s_2\} P_k^+ && (16)\\
& x'_k, x'_{k+1} \in Z^+ && (17)
\end{aligned}
$$

In fact, for any pair (x'_k, x'_{k+1}) that satisfies (15), (16) and (17) we have that

$$
x_k \leq x'_k + x'_{k+1}.
$$

Hence, u_k can be found by taking the minimum $x'_k + x'_{k+1}$ that satisfies

$$
x'_k + x'_{k+1} \geq \max\left\{ \frac{\max\{s_1, s_2\} P_k^+}{p_2 x'_{k+1}} + x'_{k+1} + \frac{s_2}{p_2}, \frac{\max\{s_1, s_2\} P_k^+}{2p_2 x'_{k+1}} + x'_{k+1} + \frac{s_1}{2p_2} \right\}.
$$

Assume that x'_k and x'_{k+1} are continuous. Considering the first term in the above equation, by the basic inequality knowledge, we know that

$$
\frac{\max\{s_1, s_2\} P_k^+}{p_2 x'_{k+1}} + x'_{k+1} + \frac{s_2}{p_2} \geq 2\sqrt{\frac{\max\{s_1, s_2\} P_k^+}{p_2}} + \frac{s_2}{p_2}
$$

and the equality holds when

$$x'_{k+1} = \sqrt{\frac{\max\{s_1, s_2\}P_k^+}{p_2}}.$$

By the value of x'_{k+1}, we obtain that

$$x'_k = \sqrt{\frac{\max\{s_1, s_2\}P_k^+}{p_2}} + \frac{s_2}{p_2}$$

from (15). Adding back the integrity of x'_k and x'_{k+1}, we construct a upper bound for the first term by

$$\left\lceil \sqrt{\frac{\max\{s_1, s_2\}P_k^+}{p_2}} \right\rceil + \left\lceil \sqrt{\frac{\max\{s_1, s_2\}P_k^+}{p_2}} + \frac{s_2}{p_2} \right\rceil. \tag{18}$$

Making the similar construction to the second term, we then have the following upper bound.

$$\begin{aligned} u_k = \max \Bigg\{ & \left\lceil \sqrt{\frac{\max\{s_1,s_2\}P_k^+}{p_2}} \right\rceil + \left\lceil \sqrt{\frac{\max\{s_1,s_2\}P_k^+}{p_2}} + \frac{s_2}{p_2} \right\rceil, \\ & \left\lceil \sqrt{\frac{\max\{s_1,s_2\}P_k^+}{2p_2}} \right\rceil + \left\lceil \sqrt{\frac{\max\{s_1,s_2\}P_k^+}{2p_2}} + \frac{s_1}{2p_2} \right\rceil \Bigg\}. \end{aligned} \tag{19}$$

The procedure to calculate the upper bounds u_i on the number of items in sublot i ($i = 1, \ldots, b$) works then as follows.

- *Step 1*. Let $P_k^+ = 0$ and $i = b$. Compute the value u_i using equation (19).
- *Step 2*. Let $P_k^+ = P_k^+ + u_i$ and $i = i - 1$. If $P_k^+ < n$, go to Step 2. Else, stop.

References

Agnetis A, Alfieri A, Nicosia G (2009) Part batching and scheduling in a flexible cell to minimize setup costs. J Sched 6:87–108

Alfieri A, Glass CA, van de Velde SL (2012) Lot streaming in a two-machine flow shop with attached setup times. IIE Trans 44:695–710

Ben-Dati L, Mosheiov G, Oron D (2009) Batch scheduling on two-machine flowshop with machine-dependent setup times. Adv Oper Res 2009, 153910

Biskup D, Feldmann M (2006) Lot streaming with variable sublots: an integer programming formulation. J Oper Res Soc 57:296–303

Brucker P, Gladky A, Hoogeveen JA, Kovalyov MY, Potts CN, Tautenhahn T, van de Velde SL (1998) Scheduling a batching machine. J Sched 1:31–54

Bukchin J, Masin M (2004) Multi-objective lot splitting for a single product m-machine flowshop line. IIE Trans 36:191–202

Bukchin J, Jaffe M, Tzur M (2002) Lot splitting to minimize average flow-time in a two-machine flow-shop. IIE Trans 34:953–970
Chang JH, Chiu HN (2005) A comprehensive review of lot streaming. Int J Prod Res 43:1515–1536
Chen J, Steiner G (1996) Lot streaming with detached setups in three-machine flow shops. Eur J Oper Res 96:591–611
Chen J, Steiner G (1998) Lot streaming with attached setups in three machine flow shops. IIE Trans 30:1075–1084
Chen J, Steiner G (1999) Discrete lot streaming in two-machine flow shops. INFOR 37:160–173
Cheng TCE, Chen ZI, Kovalyov MY, Lin BMT (1996) Parallel-machine batching and scheduling to minimize total completion time. IIE Trans 28:953–956
Cheng TCE, Lin BMT, Toker A (2000) Makespan minimization in the two-machine flowshop batch scheduling problem. Nav Res Logist 47:128–144
Cheng M, Mukherjee NJ, Sarin SC (2013) A review of lot streaming. Int J Prod Res 51:7023–7046
Cheng M, Sarin SC, Singh S (2016) Two-stage, single-lot, lot streaming problem for a 1+2 hybrid flow shop. J Global Optim 66:263–290
Congram RK, Potts CN, van de Velde SL (2002) An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. INFORMS J Comput 14:52–67
Defersha FM, Chen M (2012) Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. Int J Prod Res 50:2331–2352
Fisher ML (1981) The lagrangian relaxation method for solving integer programming problems. Manag Sci 27:1–18
Fisher ML (1985) An application oriented guide to lagrangian relaxation. Interfaces 15:10–21
Geoffrion AM (1974) Lagrangian relaxation and its uses in integer programming. Math Progr Study 2:82–114
Glass CA, Potts CN (1998) Structural properties of lot streaming in a flow shop. Math Oper Res 23:624–639
Glass CA, Gupta JND, Potts CN (1994) Lot streaming in three-stage production processes. Eur J Oper Res 75:378–394
Held M, Karp RM (1970) The traveling salesman problem and minimum spanning trees. Oper Res 18:1138–1162
Held M, Karp RM (1971) The traveling salesman problem and minimum spanning trees: part ii. Math Program 1:6–25
Held M, Wolfe P, Crowder H (1974) Validation of subgradient optimization. Math Program 6:62–88
Kalir AA, Sarin SC (2001) Optimal solution for the single batch flow-shop lot streaming problem with equal sublots. Decis Sci 32:387–397
Kalir AA, Sarin SC (2003) Constructing near optimal schedules for the flow-shop lot streaming problem with sublot-attached setups. J Combin Optim 7:23–44
Kalir AA, Sarin SC (2008) A single-lot, unified cost-based flow shop lot-streaming problem. Int J Prod Econ 113:413–424
Mosheiov G, Oron D, Ritov Y (2004) Flow-shop batch scheduling with identical processing-time jobs. Nav Res Logist 51:783–799
Potts CN, Baker KR (1989) Flow shop scheduling with lot streaming. Oper Res Lett 8:297–303
Santos C, Magazine M (1985) Batching in single operation manufacturing systems. Oper Res Lett 4:99–103
Shapiro JF (1979) A survey of lagrangian techniques for discrete optimization. Ann Discret Math 5:113–138
Silver EA, Pyke DF, Peterson R (1998) Inventory management and production planning and scheduling. Wiley, New York
Trietsch D (1987) Optimal transfer lots for batch manufacturing: a base case and extensions. Technical report nps-54-89-011, Naval Postgraduate School, Monterey, California
Trietsch D, Baker KR (1993) Basic techniques for lot streaming. Oper Res 41:1065–1076
Xuan H, Tang LX (2007) Scheduling a hybrid flowshop with batch production at the last stage. Comput Oper Res 34:2718–2733

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