An efficient implementation of a VNS heuristic for the weighted fair sequences problem

C. Rocha, B. J.S. Pessoa, D. Aloise, L. A. Cabral

An efficient implementation of a VNS heuristic for the weighted fair sequences problem

Caroline Rocha [a, b]

Bruno J.S. Pessoa [c]

Daniel Aloise [a, d]

Lucidio A. Cabral [c]

[a] GERAD, Montréal (Qc), Canada, H3T 1J4

[b] School of Sciences and Technology, Universidade Federal do Rio Grande do Norte, 59078–970, Natal, Brazil

[c] Centro de Informática, Universidade Federal da Paraíba, 58055–000, João Pessoa, Brazil

[d] Département de génie informatique et génie logiciel,, Polytechnique Montréal, Montréal (Québec), Canada H3C 3A7

caroline.rocha@ect.ufrn.br
bruno@ci.ufpb.br
daniel.aloise@polymtl.ca
lucidio@ci.ufpb.br

Abstract : In the Weighted Fair Sequences Problem (WFSP), one aims to schedule a set of tasks or activities so that the maximum product between the largest temporal distance between two consecutive executions of a task and its priority is minimized. The WFSP covers a large number of applications in different areas, ranging from automobile production on a mixed-model assembly line to the sequencing of interactive applications to be aired in a Digital TV environment. This paper proposes an iterative heuristic method for the WFSP centered on an efficient implementation of a variable neighborhood search heuristic. Computational experiments on benchmark instances show that the proposed metaheuristic outperforms the state-of-the-art method proposed to the problem, obtaining comparable solution values in much less computational time.

Keywords: Scheduling, fair sequences, heuristics, metaheuristics

1   Introduction

Scheduling problems deal with the decision-making process regarding the allocation of limited resources to tasks over given time periods (Pinedo, 2008). Due to its large number of practical applications, scheduling problems have been tackled in different ways in the literature, giving rise to several branches of study that vary according to the goal to be achieved. Among such branches, some of them focus on the equitable sharing of resources over time. They cover numerous applications, ranging from task scheduling in real-time systems to automobile production on a mixed-model assembly line. Toyota was one of the first companies to realize that producing cars at a uniform rate over time minimize storage costs and machine idleness (Kubiak, 2004). To that end, the company created the Just-in-Time production system (Dhamala and Kubiak, 2005). Scheduling problems where the temporal distances between successive executions of tasks or activities should be within pre-specified values, in order to promote the fair use of resources, belong to a class of problems referred to in the literature as fair sequences. Pessoa et al. (2018) have recently introduced a NP-hard optimization problem named the Weighted Fair Sequences Problem (WFSP), which is a member of the above-mentioned class of problems. In the WFSP, the higher the priority of a task, the smaller and evenly-spaced should be the distances between its consecutive occurrences.

The WFSP emcompasses various practical applications, as, for instance, the scheduling of commercial advertisements in the television industry. Bollapragada et al. (2004) studied a problem faced by the National Broadcasting Company (NBC), one of the most important companies in the US television industry, and described the relationship between the company and its clients. Major advertisers buy large quantities of time slots in order to air their advertisements and require that they are spaced as regularly as possible. In the spirit of this work, García-Villoria and Salhi (2014) addressed an optimization problem that aims to find a feasible broadcast scheduling that minimizes the irregularity of the interval times between airings of the same advertisement while taking into account audience rating constraints.

The WFSP also occurs in Digital TV systems in which smart advertisements are aired by means of interactive TV applications. These computer programs are cyclically aired, from servers to clients on a one-way communication channel (Morris, 2005). Therefore, to download them, the user terminals should listen to the communication channel until the arrival of the expected data. Since the advertisers require that the users do not wait too long for their applications, the objective is to minimize the maximum temporal distance between successive transmissions of the same smart advertisement. It is noteworthy to mention that each ad has a different priority, which varies according to the amount paid for its broadcasting.

Periodic machine maintenance (Anily et al., 1998; Bar-Noy et al., 2002) is another typical application of the WFSP. As the maintenance costs increase proportionally with time since the last maintenance, the goal is to build a scheduling of maintenance services that minimizes the temporal distance between two consecutive maintenance operations carried out on the same machine or station.

Similarly, the WFSP might be applied to the problem of scheduling physicians in an Intensive Care Unit (ICU) (Erhard et al., 2018). As some hospitals hire physicians under individual contracts, the number of shifts worked per month might vary from one doctor to another. If the physicians concentrate all their shifts in a single week, the temporal distances between two consecutive shifts might become too short, not satisfying labor regulations. Conversely, if it takes too long from one shift to another, that might cause a disruption in the treatment of the patients. The objective is to schedule shifts in a way that the temporal distances between consecutive shifts of a doctor are as constant as possible.

In the literature, problems related to the WFSP have been introduced over time in different contexts such as satellite communication (Holte et al., 1989), data broadcast (Acharya et al., 1995; Bar-Noy and Ladner, 2003), and task scheduling in real-time systems (Waldspurger and Weihl. W., 1995; Han et al., 1996).

In addition to proposing a mathematical optimization formulation for the WFSP based on mixed-integer linear programming, Pessoa et al. (2018) proved that the WFSP is NP-hard and proposed an heuristic method that iteratively solves a constrained version of the WFSP. More recently, Sinnl (2021) developed an iterative exact method that decomposes the WFSP in a series of constrained subproblems. By means of a series of new valid inequalities, the author was able to considerably increase the number of WFSP instances solved up to optimality. Our work proposes the first metaheuristic solution approach for the WFSP and compares its performance with the method of Sinnl (2021), both in terms of solution quality and efficiency.

The remainder of the paper is organized as follows. Section 2 presents a formal definition for the WFSP, including its mathematical programming formulation. Section 3 presents our iterative heuristic method along with its solution representation and specialized data structures. Section 4 contains the computational results. Finally, Section 5 provides some concluding remarks.

2   The Weighted Fair Sequences Problem (WFSP)

The WFSP can be formulated as follows. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n symbols to be sequenced. Each of them have a priority $c_i = c(x_i)$, which is determined by a priority function $c : X \to \mathbb{Z}^+$. Let S be a solution to the problem, consisting of a circular sequence of $T \leq TMAX$ symbols, where $TMAX$ is a parameter denoting the maximum length of the feasible sequences. A symbol may appear in the sequence more than once in accordance with its priority so that the ideal number of occurrences or copies of a symbol is a variable of the problem. Assuming that D_i is the largest distance found between consecutive copies of symbol x_i, including the distance between the last copy of x_i in a cycle and its first copy in the subsequent cycle, the aim is to find a solution sequence S that minimizes the largest product $P \geq D_i c_i$, for $i = 1, \ldots, n$, subject to $T \leq TMAX$.

To illustrate the main components of the problem, consider a WFSP instance for $X = \{A, B, C, D, E\}$ and $TMAX = 9$, with priorities given by $c_1 = 10, c_2 = 6, c_3 = 4, c_4 = 2$, and $c_5 = 1$. The trivial solution (i.e., the one for which each symbol has only one copy) is depicted in the left part of Figure 1, for which the largest product is $P = D_1 c_1 = 50$. The best solution is however shown in the right part of figure, where $D_1 c_1 = 30$, $D_2 c_2 = 30$, $D_3 c_3 = 24$, $D_4 c_4 = 20$, and $D_5 c_5 = 10$. As we can see, P decreases from 50 to 30 by comparing the two solutions.
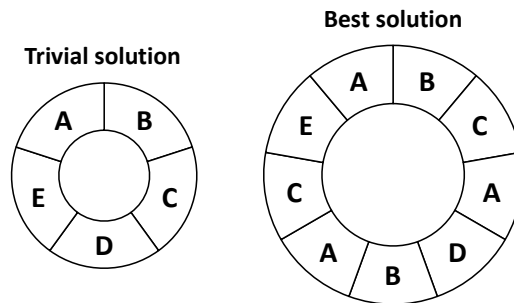
Figure 1: Two different WFSP solutions to the set of symbols $X = \{A, B, C, D, E\}$.

A mixed-integer linear formulation for the WFSP was provided by Pessoa et al. (2018) with a series of valid cuts. For example, the authors proved that there exists an optimal WFSP solution such that no consecutive copies of a symbol occur in consecutive positions of the sequence. The authors still included a series of cuts to break symmetry. Since the sequence is circular it can actually start from any position, thus yielding multiple equivalent solutions. More recently, Sinnl (2021) proposed to solve the WFSP by solving a sequence of subproblems in which the sequence length is fixed. From that variable fixing, the author demonstrates a series of valid inequalities for those subproblems, allowing his algorithm to increase the number of WFSP benchmark instances solved up to optimality.

3 Iterative algorithm for the WFSP

As done by Sinnl (2021), the general outline of our heuristic, presented in Algorithm 1, decomposes the WFSP into a series of subproblems of fixed sequence length $T = n, \ldots, TMAX$.

Algorithm 1 Iterative solution heuristic for the WFSP

```
 1: procedure ITERATIVE
 2:     S* ← trivial WFSP solution with |S| = n
 3:     repeat
 4:         S ← INCREASESEQUENCE(S)
 5:         S ← VNS(S)
 6:         if cost(S) < cost(S*) then
 7:             S* ← S
 8:         end if
 9:     until |S| = TMAX
10:     return S*
11: end procedure
```

The algorithm starts from a trivial WFSP solution sequence containing one single copy of each symbol $x_i \in X$, which is initially stored as the best solution sequence S^* found so far. Then, it proceeds in line 4 by increasing the solution sequence through the addition of a new copy of a symbol from X in the best possible position (Section 3.2). In the sequel, a general Variable Neighborhood Search (VNS) is called in line 5 to approach the current WFSP subproblem with fixed sequence length $|S|$ (Section 3.1). In case the obtained solution is better than the best one found so far in line 6, S^* is updated in line 7. The loop of lines 3-9 is repeated until the subproblem with fixed length equal to $TMAX$ has been optimized. Finally, the best WFSP solution S^* is returned in line 10.

3.1 VNS for the WFSP

In this section, we provide more details about the VNS heuristic (Mladenović and Hansen, 1997) used within our iterative solution heuristic. It follows the precepts of the recent *less is more approach* (LIMA) in metaheuristic design (Mladenović et al., 2016), which advocates the use of a minimal number of algorithmic ingredients (e.g. number of local searches, data reduction, decomposition, etc.) as a dimension for metaheuristic evaluation. In fact, VNS heuristics based on a unique neighborhood structure might be very effective (e.g. (Costa et al., 2017; Hansen et al., 2012; Mladenović et al., 2020)) In the LIMA spirit, we exploit in this work two neighbourhoods for the WFSP in a very efficient way.

3.1.1 Specialized data structures

Our heuristic maintains a set of data structures to support the efficient exploration of the WFSP solution space. They are used within the VNS local searches in order to speed up the most time consuming part of the proposed heuristic.

We represent a solution for the WFSP by an one-dimensional array S that stores a sequence $(s_1, s_2, \ldots, s_T)$ of T symbols of X. From S, auxiliary information can be computed and stored in additional data structures as listed below:

- $\Gamma^+[x_i, j]$: position of the first copy of symbol x_i which appears after position j;
- $\Gamma^-[x_i, j]$: position of the first copy of symbol x_i which appears before position j;
- $d(j)$: distance between the copy of symbol s_j at position j and the next copy of s_j in S, that is, $d(j) = \Gamma^+[s_j, j] - j$, if $\Gamma^+[s_j, j] \geq j$; otherwise, $d(j) = \Gamma^+[s_j, j] - j + T$;
- $D(x_i)$: maximum distance found among all pairs of consecutive copies of symbol $x_i \in X$ in S;
- $p(j)$: product $d(j) \times c(s_j)$ associated with the copy of symbol s_j at position j;
- $m(x_i)$: number of copies of symbol $x_i \in X$ in S;
- $cost(S)$: cost of the sequence S, that is, $cost(S) = \max\{D(x_i)|x_i \in X\}$.

In Figure 2, we illustrate a WFSP solution of length 13 for the set of symbols $X = \{x_1, x_2, x_3, x_4\} = \{A, B, C, D\}$ for which $c(x_i) = 1, \forall x_i \in X$. It also presents the content of the matrices Γ^+ and Γ^- as well as the content of the array d associated with that solution.

j	1	2	3	4	5	6	7	8	9	10	11	12	13
	A	B	A	C	D	B	A	D	B	A	C	B	D

	1	2	3	4	5	6	7	8	9	10	11	12	13
$d(j)$	2	4	4	7	3	3	3	5	3	4	6	3	5
$\Gamma^+[A,j]$	3	3	7	7	7	7	10	10	10	1	1	1	1
$\Gamma^+[B,j]$	2	6	6	6	6	9	9	9	12	12	12	2	2
$\Gamma^+[C,j]$	4	4	4	11	11	11	11	11	11	11	4	4	4
$\Gamma^+[D,j]$	5	5	5	5	8	8	8	13	13	13	13	13	5
$\Gamma^-[A,j]$	10	1	1	3	3	3	3	7	7	7	10	10	10
$\Gamma^-[B,j]$	12	12	2	2	2	2	6	6	6	9	9	9	12
$\Gamma^-[C,j]$	11	11	11	11	4	4	4	4	4	4	4	11	11
$\Gamma^-[D,j]$	13	13	13	13	13	5	5	5	8	8	8	8	8

Figure 2: Example of a sequence solution and its data structures d, Γ^+ and Γ^-.

We also compute from the sequence S its *dominant interval*, which correspond to the interval of positions in the sequence associated with its maximum product. There might exist several dominant intervals in a sequence, potentially associated to different symbols. We note in Figure 2 that the dominant interval for the given example is $[4, 11]$, which is responsible for the maximum distance found in the given sequence between the copies of the symbol $x_3 = C$.

An important feature of our procedures is that they focus the search for improving solutions in the dominant intervals of S. Indeed, since our objective function is given by the minimum of maximum products $D_i \times c_i$, for $i = 1, \ldots, n$, only products associated to dominant intervals of S are in fact active. Our specialized data structures are also designed in a way that updates are performed only for the indices associated with the modified interval. Figure 3 presents a solution for which the symbol $x_4 = D$ at position 8 is replaced by a symbol $x_3 = C$. The figure shows the content of the matrices Γ^+ and Γ^- as well as the array d associated with such modification. We note that only the values of the shaded cells in the figure need to be updated.

In general, the updates in the neighborhoods explained in the next sections are performed in time $O(n \times TMAX)$. However, this time is largely mitigated by the fast evaluation of neighboring solutions inside our VNS, which is performed in constant time as will be discussed next.

3.1.2   Neighborhood structures

The following neighborhood structures are used within our VNS heuristic:

- *Flip* ($\mathcal{N}_1$) – A copy of a symbol is changed to that of another symbol; and
- *Shift-one* ($\mathcal{N}_2$) – A copy of a symbol is shifted to the next position in the sequence.

3.1.3   Flip neighborhood

From a solution sequence S, the flip neighborhood encompasses all the neighboring solutions obtained by changing the symbol s_j at position j by a symbol $x_i \in X \setminus \{s_j\}$, denoted here $\textsc{Flip}(S, j, x_i)$.

j	1	2	3	4	5	6	7	8	9	10	11	12	13
	A	B	A	C	D	B	A	C	B	A	C	B	D

	1	2	3	4	5	6	7	8	9	10	11	12	13
$d(j)$	2	4	4	4	8	3	3	3	3	4	6	3	5
$\Gamma^+[A,j]$	3	3	7	7	7	7	10	10	10	1	1	1	1
$\Gamma^+[B,j]$	2	6	6	6	6	9	9	9	12	12	12	2	2
$\Gamma^+[C,j]$	4	4	4	8	8	8	8	11	11	11	4	4	4
$\Gamma^+[D,j]$	5	5	5	5	13	13	13	13	13	13	13	13	5
$\Gamma^-[A,j]$	10	1	1	3	3	3	3	7	7	7	10	10	10
$\Gamma^-[B,j]$	12	12	2	2	2	2	6	6	6	9	9	9	12
$\Gamma^-[C,j]$	11	11	11	11	4	4	4	4	8	8	8	11	11
$\Gamma^-[D,j]$	13	13	13	13	13	5	5	5	5	5	5	5	8

Figure 3: Update of data structures d, Γ^+ and Γ^- after s_8 is changed.

We note that the evaluation within this neighborhood involves only two symbols: x_i and s_j. As such, a neighbor solution in the flip neighborhood may be better than the current solution only if:

$$p(\Gamma^-[x_i,j]) > p(\Gamma^-[s_j,j]) + p(j), \tag{1}$$

that is, the current product associated with the entering symbol x_i at position $\Gamma^-[x_i,j]$ (on the left-hand side) must be greater than the resulting product associated with exiting symbol s_j from position j (on the right-hand side). If that is not the case, the flip breaks an interval between consecutive copies of x_i which is actually less expensive than the new resulting interval between copies of s_j.

To illustrate, let us revisit the example in Figure 2, now assuming that the priorities are $c_1 = 10$, $c_2 = 8$, $c_3 = 6$, and $c_4 = 3$. Let us consider switching the symbol D at position 8 by $x_3 = C$. This would decrease the cost of the solution from 42 to 40 – the new dominant interval becomes associated to A. We note that inequality (1) holds since:

$$p(\Gamma^-[C,8]) > p(\Gamma^-[D,8]) + p(8)$$
$$p(4) > p(5) + p(8)$$
$$42 > 9 + 15.$$

An important observation is that a neighbor solution obtained in the flip neighborhood can improve the cost of a solution S if and only if

$$p(\Gamma^-(x_i,j)) = D(x_i) = cost(S),$$

that is, if the entering symbol x_i is associated to the dominant interval of S. Thus, the dominant interval is broken into two new intervals for x_i. This fact is largely exploited in our algorithm to significantly reduce the number of move evaluations performed during the local search.

3.1.4 Shift-one neighborhood

This neighborhood contains all neighboring solutions obtained from S by shifting the copy of the symbol located at position j to the position $((j+1) \mod T)$. Note that the shift-one neighborhood encompasses the application of two consecutive flip moves: $\text{FLIP}(S, j+1, s_j)$ and $\text{FLIP}(S, j, s_{j+1})$, being denoted here $\text{SHIFTONE}(S, j)$. Therefore, its evaluation can also be performed in constant time.

Precisely, a neighbor in the shift-one neighborhood may be better than the current solution only if:

$$\max \left\{ p(\Gamma^-[s_j, j]), p(\Gamma^-[s_{j+1}, j+1]), p(j), p(j+1) \right\} >$$
$$\max \left\{ (p(\Gamma^-[s_j, j]) + c(s_j), p(\Gamma^-[s_{j+1}, j+1]) - c(s_{j+1}), \right.$$
$$\left. p(j) - c(s_j), p(j+1) + c(s_{j+1}) \right\}. \tag{2}$$

The shift of a symbol located at position j affects the data structures of four sequence locations: $\Gamma^-[s_j, j]$, $\Gamma^-[s_{j+1}, j+1]$, j, and $j+1$. Thus, inequality (2) computes the maximum product $p(\cdot)$ involving the affected intervals before (in the left) and after (in the right) a shift is applied at position j. Indeed, the neighboring solution can be better than the current sequence S only if that maximum product is reduced after the shift.

Referring back to the example in Figure 2 with priorities $c_1 = 10$, $c_2 = 8$, $c_3 = 6$, and $c_4 = 3$, the neighbor yielded by SHIFTONE$(S, 4)$ satisfies the inequality (2) as follows:

$$\max \left\{ p(\Gamma^-[C, 4]), p(\Gamma^-[D, 5]), p(4), p(5) \right\} >$$
$$\max \left\{ (p(\Gamma^-[C, 4]) + c(C), p(\Gamma^-[D, 5]) - c(D), \right.$$
$$\left. p(4) - c(C), p(5) + c(D) \right\},$$

i.e.,

$$\max \left\{ p(11), p(13), p(4), p(5) \right\} > \max \left\{ p(11) + 6, p(13) - 3, p(4) - 6, p(5) + 3 \right\}$$
$$42 > 36.$$

Observe that the cost of the resulting WFSP solution is however decreased from 42 to 40, due to the fact that the new incumbent dominant interval in the sequence is now associated with symbol A.

3.1.5 Shaking

In the VNS framework, a shaking step is typically used so that the search could escape from local minima. Our shaking step makes uses of the flip neighborhood only. Thus, the algorithm randomly selects a number of positions in the sequence to flip their symbols, making sure that at least one copy of each symbol is maintained in the sequence. We denote by SHAKE(S,k) a shaking procedure of order k applied to a solution S, where $k \in \{1, \ldots, k_{max}\}$ is the number of symbols fliped in the sequence.

3.1.6 Local search

The local descent used in our VNS employs both neighboorhoods flip $(\mathcal{N}_1)$ and shift-one $(\mathcal{N}_2)$ in a Variable Neighborhood Descent (VND) scheme as described by Algorithm 2. Given an input solution S, the algorithm first attempts to optimize the current solution using a first improving local search within neighborhood structure $\mathcal{N}_1$. Subsequently, a first improving local search within neighborhood $\mathcal{N}_2$ is used to optimize the $\mathcal{N}_1$ local optimum. If a better solution is found, the process is repeated; otherwise, a local optimum has been found for both $\mathcal{N}_1$ and $\mathcal{N}_2$ neighborhood structures.

Our local descent method accepts improvements either in terms of cost or in terms of response time variability (Corominas et al., 2008), a metric that captures information on how the copies are spread along the sequence, being expressed as:

$$\sum_{j=1}^{|S|} \left| \frac{|S|}{m(s_j)} - d(j) \right|.$$

Well distributed sequences have low response time variability. By using this metric to accept equal cost solutions, we allow the algorithm to explore larger regions of the solution space with the benefit of capturing improvements to the uniformity of the symbols copies along the sequence. Remark that this strategy also avoids the exploration of symmetric WFSP solutions.

Algorithm 2 VND algorithm

```
 1: procedure VND(S)
 2:     S′ ← S
 3:     repeat
 4:         S ← S′
 5:         S′ ← FIRSTIMPROVINGFLIP(S′)
 6:         S′ ← FIRSTIMPROVINGSHIFTONE(S′)
 7:     until S′ is not better than S
 8:     return S
 9: end procedure
```

Algorithm 3 presents the outline of the VNS algorithm used within our iterative solution heuristic (Algorithm 1).

Algorithm 3 VNS algorithm

```
 1: procedure VNS(S)
 2:     repeat
 3:         it ← 1
 4:         k ← 1
 5:         while k ≤ k_max do
 6:             S′ ← SHAKE(S, k)
 7:             S′ ← VND(S′)
 8:             if cost(S′) < cost(S) then
 9:                 S ← S′
10:                 k ← 1
11:             else
12:                 k ← k + 1
13:             end if
14:         end while
15:         it ← it + 1
16:     until it = it_max
17:     return S
18: end procedure
```

3.2   Increasing the sequence length

After each iteration of Algorithm 1, the sequence length of S must be increased so that the VNS could explore the solution space of a new WFSP subproblem. This is achieved in our algorithm by means of procedure INCREASESEQUENCE, which searches for the best possible addition of one copy of a symbol $x_i \in X$ for which $D(x_i) = cost(S)$, and that restricted to a position belonging to the associated dominant interval of S.

The addition of a symbol in the sequence solution affects the information of all symbols since each one of them has one pair of consecutive copies moved one position away – except for the entering symbol, for which a copy is inserted in the sequence. Consequently, the search for the best position for inserting a copy of a symbol is more computationally expensive than evaluating improving neighboring solutions within the flip and shift-one neighborhoods. Nonetheless, this is not critical to the performance of our iterative heuristic since only one symbol addition is made by iteration.

Very recently, Sinnl (2021) proposed a valid cut for sequence lengths $t \in \{n, \dots, TMAX\}$ from a feasible WFSP solution S. Let

$$k_i^*(S, t) = \min\{k_i : c(x_i)\lceil t/k_i \rceil < cost(S); k_i \in \mathbb{N}\}$$

be the minimum number of copies of $x_i \in X$ required to yield a solution of cost smaller than $cost(S)$ for $|S| = t$. This value is obtained by evenly distributing the copies of x_i with distance equal to $\lceil t/k_i \rceil$ along the WFSP sequence

Thus, $K^*(S, t) = \sum_{i=1}^{n} k_i^*(S, t)$ constitutes a lower bound in the number of necessary symbols to achieve a solution of cost smaller than that of S for $|S| = t$. Indeed, if $t < K^*(S, t)$, the optimal

WFSP solution is known to be of length greater than t. Consequently, the WFSP subproblem for which $|S| = t$ can be simply ignored by our iterative algorithm. The VNS procedure can be stopped as well whenever that condition is verified for a new obtained solution.

If a sequence length t is cut, the procedure INCREASESEQUENCE is applied again in order to increase the length of the current sequence via the addition of a new copy of a symbol. This is repeated until $t \geq K^*(S, t)$ or that $t > TMAX$.

Finally, we emphasize the following two facts that derive from the work of Sinnl (2021):

(i) $K^*(\cdot)$ can only increase in case a new improving WFSP solution is obtained.
(ii) If $K^*(S, t) > TMAX$, then S is *optimal*.

Fact (i) encourages the search towards good upper bounds as quick as possible in order to cut non-optimal sequence lengths. Fact (ii) provides a condition to certify an WFSP solution as optimal.

4 Computational experiments

This section reports the results obtained by our iterative heuristic. Our algorithm was implemented in C++, and the experiments were conducted on an Intel Core I5 with 2.5 GHz and 8 GB of RAM memory, running Ubuntu 16.04 LTS operation system.

For each of the 180 instances, ITERATIVE was run 10 times using $k_{max} = \lceil n/3 \rceil$ and $it_{max} = 250t$, for $t = n, \ldots, TMAX$, which means that more VNS iterations are performed for WFSP subproblems defined on larger sequence lengths. These parameter values were obtained by preliminary computational tests.

Besides the procedure ITERATIVE presented in Algorithm 1, we tested a two-pass algorithm, denoted ITERATIVE-2, for which a first pass calls ITERATIVE for a considerable smaller number of VNS iterations by using $it_{max} = t$, for $t = n, \ldots, TMAX$. This strategy is intended to lead ITERATIVE-2 towards good upper bound solutions more quickly. Then, in its second pass, ITERATIVE-2 exploits each one of the uncut WFSP subproblems using more VNS iterations, by making $it_{max} = 250t$.

Our heuristic methods ITERATIVE and ITERATIVE-2 are compared against the state-of-the-art method of Sinnl (2021) in the 440 instances of Pessoa et al. (2018) available at https://sites.google.com/site/weightedfairsequencesproblem/instances. These instances are categorized in 44 classes according to the number of symbols n and the maximum sequence length $TMAX$. Each class contains 10 distinct instances, which are generated by assigning random priorities to each symbol in the interval $[1, 2n]$. The name of the classes follows the format: n_TMAX. For example, 5_15 denotes the class with 5 symbols and $TMAX = 15$.

Table 1 presents aggregated results obtained by the algorithms in each of the instance classes. Detailed results are made available in at https://sites.google.com/site/weightedfairsequencesproblem/results. Concerning algorithms ITERATIVE and ITERATIVE-2, average and best solution values are collected from 10 distinct executions, in each one of the instances of a class. The reported aggregated results correspond to average values computed over the 10 instances of each class. We remark that the method of Sinnl (2021) is not exact if it is halted before solving all of the WFSP subproblems. In that case, the returned solution is not optimal, thus consisting of an upper bound only. That method was allowed to run for 1800 seconds for instance classes up to 15_60 in the table. After that, the method of was allowed to run for 3600 seconds.

We observe from Table 1 that in the vast majority of the classes ($\approx 82\%$) the algorithms obtain the same results. The lines in bold correspond to the 8 instance classes for which the results obtained by the tested methods differ. In two of these classes, i.e., 15_60 and 50_100, our heuristics obtained the best results, while in two of them, i.e. 13_52 and 7_63, the best results were found by the method of Sinnl (2021).

Table 1: Solution cost results for the instances of Pessoa et al. (2018).

Instance class	Sinnl (2021) cost	ITERATIVE avg. cost	ITERATIVE best cost	ITERATIVE-2 avg. cost	ITERATIVE-2 best cost
5_10	37.50	37.50	37.50	37.50	37.50
5_15	31.80	31.80	31.80	31.80	31.80
5_20	34.70	34.70	34.70	34.70	34.70
7_14	58.60	58.60	58.60	58.60	58.60
7_21	64.90	64.90	64.90	64.90	64.90
7_28	65.30	65.30	65.30	65.30	65.30
9_18	111.20	111.20	111.20	111.20	111.20
9_27	96.30	96.30	96.30	96.30	96.30
9_36	93.60	93.60	93.60	93.60	93.60
11_22	163.70	163.70	163.70	163.70	163.70
11_33	137.20	137.20	137.20	137.20	137.20
11_44	**141.70**	**141.82**	**141.80**	**141.80**	**141.70**
13_26	213.80	213.80	213.80	213.80	213.80
13_39	213.50	213.50	213.50	213.50	213.50
13_52	**176.90**	**177.50**	**177.50**	**177.53**	**177.50**
15_30	322.80	322.80	322.80	322.80	322.80
15_45	295.60	295.60	295.60	295.60	295.60
15_60	**269.00**	**269.96**	**268.60**	**269.53**	**268.60**
5_25	34.30	34.30	34.30	34.30	34.30
5_30	34.70	34.70	34.70	34.70	34.70
5_35	32.20	32.20	32.20	32.20	32.20
5_40	33.00	33.00	33.00	33.00	33.00
5_50	33.30	33.30	33.30	33.30	33.30
5_75	35.60	35.60	35.60	35.60	35.60
5_100	31.90	31.90	31.90	31.90	31.90
5_125	33.90	33.90	33.90	33.90	33.90
5_150	31.20	31.20	31.20	31.20	31.20
5_200	36.00	36.00	36.00	36.00	36.00
7_35	62.40	62.40	62.40	62.40	62.40
7_42	67.80	67.80	67.80	67.80	67.80
7_49	**64.80**	**65.22**	**64.80**	**65.00**	**64.80**
7_56	60.80	60.80	60.80	60.80	60.80
7_63	**58.90**	**59.21**	**59.00**	**59.19**	**59.00**
9_45	**102.80**	**102.88**	**102.80**	**102.90**	**102.80**
11_55	**135.00**	**135.65**	**135.00**	**135.71**	**135.30**
20_40	533.80	533.80	533.80	533.80	533.80
20_60	476.00	476.00	476.00	476.00	476.00
25_50	807.00	807.00	807.00	807.00	807.00
25_75	766.70	766.70	766.70	766.70	766.70
30_60	1179.80	1179.80	1179.80	1179.80	1179.80
35_70	1569.10	1569.10	1569.10	1569.10	1569.10
40_80	2121.90	2121.90	2121.90	2121.90	2121.90
45_90	2601.70	2601.70	2601.70	2601.70	2601.70
50_100	**3397.80**	**3362.20**	**3362.20**	**3362.20**	**3362.20**

The results of Table 2 allow us to better demonstrate the advantages of our proposed heuristics over the current state-of-the-art method for the WFSP. In that table, we report the average computing times spent by the tested algorithms in each instance class. Besides, we report in the table the number of verified optimal solutions (column #*nopt*) obtained by each algorithm. The optimality certificates are provided either by the method of Sinnl (2021), or by fact (ii) (see Section 3.2). We conclude from Table 2 that:

- The proposed heuristics ITERATIVE and ITERATIVE-2 are as effective as the method of Sinnl (2021) in terms of WFSP solutions solved up to optimality. They are actually slightly more efficient (409 vs. 405 instances optimally solved), being two times more effective when solving the largest instances of the data set, i.e., class 50_100. We note that it might exist other solutions which are in fact optimal. Nonetheless, we cannot prove their optimality.

Table 2: CPU times and number of optimal solutions per instance class.

Instance	Sinnl (2021)		ITERATIVE		ITERATIVE-2	
class	time (sec.)	#nopt	avg time (sec.)	#nopt	avg Time (sec.)	#nopt
05_10	0.04	10	0.01	10	0.00	10
05_15	0.10	10	0.04	10	0.02	10
05_20	0.31	10	0.15	10	0.12	10
07_14	0.11	10	0.01	10	0.00	10
07_21	0.55	10	0.17	10	0.13	10
07_28	5.71	10	0.48	10	0.44	10
09_18	0.24	10	0.04	10	0.00	10
09_27	0.91	10	0.06	10	0.02	10
09_36	13.13	10	0.60	10	0.42	10
11_22	0.53	10	0.09	10	0.00	10
11_33	2.89	10	0.11	10	0.00	10
11_44	189.40	10	0.88	9	0.75	10
13_26	1.33	10	0.20	10	0.00	10
13_39	10.26	10	0.11	10	0.00	10
13_52	390.60	8	2.18	7	1.58	7
15_30	3.05	10	0.23	10	0.00	10
15_45	30.78	10	0.14	10	0.00	10
15_60	377.60	9	2.34	10	1.76	10
05_25	2.35	10	0.62	10	0.53	10
05_30	3.05	10	0.61	10	0.51	10
05_35	10.98	10	1.21	10	1.11	10
05_40	173.92	10	2.15	10	2.09	10
05_50	407.71	10	3.88	10	3.96	10
05_75	837.11	8	14.04	8	14.71	8
05_100	450.99	9	29.08	9	30.93	9
05_125	1518.29	6	87.13	6	95.78	6
05_150	1618.53	6	165.13	6	193.86	6
05_200	2186.13	4	376.50	4	444.06	4
07_35	79.36	10	1.18	10	1.05	10
07_42	91.21	10	2.06	10	1.92	10
07_49	566.62	9	3.54	9	3.38	9
07_56	798.94	8	5.97	8	6.13	8
07_63	1595.93	6	14.89	6	15.30	6
09_45	802.45	8	1.86	8	1.51	8
11_55	674.43	9	5.44	9	5.12	8
20_40	11.57	10	0.54	10	0.00	10
20_60	116.78	10	0.65	10	0.02	10
25_50	49.27	10	1.09	10	0.01	10
25_75	763.45	10	3.00	10	0.12	10
30_60	140.12	10	3.59	10	0.03	10
35_70	383.09	10	10.42	10	0.08	10
40_80	763.53	10	7.58	10	0.13	10
45_90	1976.25	10	17.56	10	0.25	10
50_100	3180.80	5	23.12	10	0.62	10
	738.57	405	**17.97**	**409**	**18.83**	**409**

- In average, the proposed heuristics are approximately 40 times more efficient than the method of Sinnl (2021). We note that our experiments are performed is a computational architecture akin to that of Sinnl (2021).

- The two-pass approach employed by the heuristic ITERATIVE-2 did not appear to be faster than the heuristic ITERATIVE. This might be because our VNS is often halted in its first iterations, after verifying that its incumbent solution cannot be improved for the current sequence length (see section 3.2).

- The most computationally intensive instances to our heuristics seem to be those with a small number of symbols n and a large maximum sequence length $TMAX$. In general, our heuristics execute for longer for instances in which they are not able to cut WFSP subproblems by means of the lower bound on the number of necessary symbols (see section 3.2). Thus, the inner VNS

is obliged to run until the maximum number of allowed VNS iterations (it_{max}) is achieved. Nonetheless, it is likely that several of the solutions obtained for these instances are also optimal.

5 Concluding remarks

We described in this work an iterative heuristic method for approximately solving the Weighted Fair Sequences Problem (WFSP) proposed by Pessoa et al. (2018). Our method is based on the systematic exploration of a sequence of subproblems for which the sequence length of the WFSP is fixed. The solution for each subproblem is then centered on a general VNS method that exploits two neighborhoods during the search. We proposed optimized data structures to speed up its local search step, and which are essential for the efficiency of the whole algorithm. The method also relies on a recent valid inequality proposed by Sinnl (2021), which allows it to ignore WFSP subproblems that cannot contain WFSP optima.

Computational experiments have shown that our heuristic is as good as the state-of-the-art method in terms of the quality of the solutions found. Nonetheless, it is considerable more efficient, being 40 times faster in average when solving benchmark instances from the literature.

References

Acharya, S., Alonso, R., Franklin, M., Zdonik, S., 1995. Broadcast disks: Data management for asymmetric communication environments. SIGMOD Rec. 24, 2, 199–210.

Anily, S., Glass, C.A., Hassin, R., 1998. The scheduling of maintenance service. Discrete Applied Mathematics 82, 1, 27–42.

Audsley, N.C., Burns, A., Davis, R.I., Tindell, K.W., Wellings, A.J., 1995. Fixed priority pre-emptive scheduling: An historical perspective. Real-Time Systems 8, 2, 173–198.

Baker, K., 1974. Introduction to Sequencing and Scheduling. Wiley.

Baker, K.R., Trietsch, D., 2009. Principles of Sequencing and Scheduling. Wiley Publishing.

Bar-Noy, A., Bhatia, R., Naor, J.S., Schieber, B., 2002. Minimizing service and operation costs of periodic scheduling. Math. Oper. Res. 27, 3, 518–544.

Bar-Noy, A., Ladner, R.E., 2003. Windows scheduling problems for broadcast systems. SIAM J. Comput. 32, 4, 1091–1113.

Baruah, S.K., 1995. Fairness in periodic real-time scheduling. In Proceedings of the 16th IEEE Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, pp. 200–.

Baruah, S.K., Cohen, N.K., Plaxton, D.A. C. G.and Varvel, 1996. Proportionate progress: A notion of fairness in resource allocation. Algorithmica 15, 6, 600–625.

Bollapragada, S., Bussieck, M.R., Mallik, S., 2004. Scheduling commercial videotapes in broadcast television. Operations Research 52, 5, 679–689.

Brucker, P., 2010. Scheduling Algorithms (5th edn.). Springer Publishing Company, Incorporated.

Chan, M.Y., Chin, F., 1993. Schedulers for larger classes of pinwheel instances. Algorithmica 9, 5, 425–462.

Cheng, S.C., Stankovic, J.A., Ramamritham, K., 1989. Tutorial: Hard real-time systems. IEEE Computer Society Press, Los Alamitos, CA, USA, chapter Scheduling Algorithms for Hard Real-time Systems: A Brief Survey, pp. 150–173.

Coppersmith, D., Leem, J., Leung, J., 1999. A polytope for a product of real linear functions in 0/1 variables. IBM Research Report RC21568.

Corominas, A., García-Villoria, A., Pastor, R., 2008. Solving the response time variability problem by means of multi-start and grasp metaheuristics. In Proceedings of the 2008 Conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence, IOS Press, Amsterdam, The Netherlands, The Netherlands, pp. 128–137.

Corominas, A., Kubiak, W., Palli, N.M., 2007. Response time variability. Journal of Scheduling 10, 2, 97–110.

Corominas, A., Kubiak, W., Pastor, R., 2010. Mathematical programming modeling of the response time variability problem. European Journal of Operational Research 200, 2, 347–357.

Costa, L.R., Aloise, D., Mladenović, N., 2017. Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering. Information Sciences 415, 247–253.

Dhamala, T.N., Kubiak, W., 2005. A brief survey of just-in-time sequencing for mixed-model systems. International Journal of Operations Research 2, 2, 38–47.

Erhard, M., Schoenfelder, J., Fügener, A., Brunner, J.O., 2018. State of the art in physician scheduling. European Journal of Operational Research 265, 1, 1–18.

García-Villoria, A., Corominas, A., Delorme, X., Dolgui, A., Kubiak, W., Pastor, R., 2013. A branch and bound algorithm for the response time variability problem. Journal of Scheduling 16, 2, 243–252.

García-Villoria, A., Pastor, R., 2010. Solving the response time variability problem by means of a genetic algorithm. European Journal of Operational Research 202, 2, 320–327.

García-Villoria, A., Pastor, R., 2013. Minimising maximum response time. Computers & Operations Research 40, 10, 2314–2321.

García-Villoria, A., Salhi, S., 2014. Scheduling commercial advertisements for television. International journal of production research 53, 4, 1198–1215.

García-Villoria, A., Pastor, R., 2013. Simulated annealing for improving the solution of the response time variability problem. International Journal of Production Research 51, 16, 4911–4920.

Han, C., Lin, K., Hou, C., 1996. Distance-constrained scheduling and its applications to real-time systems. IEEE Trans. Computers 45, 7, 814–826.

Hansen, P., Mladenović, N., 2003. Variable Neighborhood Search, Springer US, Boston, MA. pp. 145–184.

Hansen, P., Mladenović, N., Moreno Pérez, J.A., 2008. Variable neighbourhood search: methods and applications. 4OR 6, 4, 319–360.

Hansen, P., Ruiz, M., Aloise, D., 2012. A vns heuristic for escaping local extrema entrapment in normalized cut clustering. Pattern recognition 45, 12, 4337–4345.

Holte, R., Mok, A., Rosier, L., Tulchinsky, I., Varvel, D., 1989. The pinwheel: A real-time scheduling problem. In 22nd Hawaii International Conference on System Sciences, Kailua-Kona, pp. 693–702.

Kenyonand, Schabanel, 2003. The data broadcast problem with non-uniform transmission times. Algorithmica 35, 2, 146–175.

Kim, E.S., Glass, C.A., 2014. Perfect periodic scheduling for three basic cycles. Journal of Scheduling 17, 1, 47–65.

Kimms, A., Muller-Bungart, M., 2007. Revenue management for broadcasting commercials: The channel's problem of selecting and scheduling the advertisements to be aired, 1, 28–44.

Kubiak, W., 1993. Minimizing variation of production rates in just-in-time systems: A survey. European Journal of Operational Research 66, 3, 259–271.

Kubiak, W., 2004. Fair sequences. In In Handbook of Scheduling: Algorithms, Models and Performance Analysis, Leung, J.Y-T., editor, Chapman & Hall/CRC, Boca.

Kubiak, W., Sethi, S., 1994. Optimal just-in-time schedules for flexible transfer lines. International Journal of Flexible Manufacturing Systems 6, 2, 137–154.

Lehoczky, J., Sha, L., Ding, Y., 1989. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In [1989] Proceedings. Real-Time Systems Symposium, pp. 166–171.

Leinbaugh, D.W., 1980. Guaranteed response times in a hard-real-time environment. IEEE Transactions on Software Engineering SE-6, 1, 85–91.

Leung, J., Kelly, L., Anderson, J.H., 2004. Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton, FL, USA.

Leung, J., Merrill, M., 1980. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. Information Processing Letters 11, 3.

Leung, J.Y.T., Whitehead, J., 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation. 2, 4, 237–250.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1, 46–61.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated Local Search, Springer US, Boston MA. pp. 320–353.

Miltenburg, J., 1989. Level schedules for mixed-model assembly lines in just-in-time production systems. Management Science 35, 2, 192–207.

Mladenović, N., Alkandari, A., Pei, J., Todosijević, R., Pardalos, P.M., 2020. Less is more approach: basic variable neighborhood search for the obnoxious p-median problem. International Transactions in Operational Research 27, 1, 480–493.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24, 11, 1097–1100.

Mladenović, N., Todosijević, R., Urošević, D., 2016. Less is more: basic variable neighborhood search for minimum differential dispersion problem. Information Sciences 326, 160–171.

Monden, Y., 1983. Toyota production system : practical approach to production management. Norcross, GA : Industrial Engineering and Management Press, Institute of Industrial Engineers. pp.233–246.

Morris, S., 2005. Interactive TV standards: a guide to MHP, OCAP, and JavaTV. Elsevier, San Diego, CA.

Pessoa, B.J.S., Aloise, D., Cabral, L.A., 2018. The weighted fair sequences problem. Computers and Operations Research 91, Supplement C, 121–131.

Pinedo, M.L., 2008. Scheduling: Theory, Algorithms, and Systems (3rd edn.). Springer Publishing Company, Incorporated.

Ramamurthy, S., Moir, M., 2000. Static-priority periodic scheduling on multiprocessors. In Proceedings 21st IEEE Real-Time Systems Symposium, pp. 69–78.

Sha, L., Abdelzaher, T., årzén, K.E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K., 2004. Real time scheduling theory: A historical perspective. Real-Time Systems 28, 2, 101–155.

Sinnl, M., 2021. An iterative exact algorithm for the weighted fair sequences problem. arXiv preprint arXiv:2108.03024

Tari, F.G., Alaei, R., 2013. Scheduling tv commercials using genetic algorithms. International Journal of Production Research 51, 16, 4921–4929.

Waldspurger, C.A., Weihl. W., E., 1995. Stride Scheduling: Deterministic Proportional- Share Resource Management. Technical report, Cambridge, MA, USA.