CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Balancing schedules using maximum leximin
Authors	Toffano, Federico;Wilson, Nic
Publication date	2019-09-04
Original Citation	Toffano, F. and Wilson, N. (2019) 'Balancing Schedules Using Maximum Leximin', ECSQARU 2019: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Belgrade, Serbia, 18-20 September, Lecture Notes in Computer Science (LNCS, volume 11726) Cham: Springer International Publishing, pp. 492-503. doi: 10.1007/978-3-030-29765-7_41
Type of publication	Conference item
Link to publisher's version	https://link.springer.com/chapter/10.1007%2F978-3-030-29765-7_41 - 10.1007/978-3-030-29765-7_41
Rights	© Springer Nature Switzerland AG 2019. This is a post-peer-review, pre-copyedit version of an article published in Lecture Notes in Computer Science The final authenticated version is available online at: http://dx.doi.org/10.1007/978-3-030-29765-7_41
Download date	2026-11-06 05:11:18
Item downloaded from	https://hdl.handle.net/10468/8794



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Balancing Schedules Using Maximum Leximin*

Federico Toffano and Nic Wilson

Insight Centre for Data Analytics, School of CS and IT
University College Cork, Cork, Ireland
{federico.toffano,nic.wilson}@insight-centre.org

Abstract. We consider the problem of assigning, in a fair way, time limits for processes in manufacturing a product, subject to a deadline where the duration of each activity can be uncertain. We focus on an approach based on choosing the maximum element according to a leximin ordering, and we prove the correctness of a simple iterative procedure for generating this maximally preferred element. Our experimental testing illustrates the efficiency of our approach.

Keywords: Fair division · Preferences · Scheduling under uncertainty

1 Introduction

We consider a network representing the manufacturing processes required to make a particular product. Each of the n edges represents one of the activities (i.e., processes) involved, and the network structure implies precedence constraints between the activities, allowing activities to be in series or in parallel. We assume an overall time limit D, and we wish to assign a time limit (i.e., maximum duration) $\text{dur}_j \geq 0$ to each activity j in a way that is consistent with the overall time limit, i.e., so that the makespan (the length of the longest path) is at most D, when the length of edge j is equal to dur_j.

Computing such time limits can be useful in a number of ways: given a delayed order, we can understand which are the activities most to blame by considering their *lateness* defined as $C_j - \text{dur}_j$, where C_j is the completion time of activity j. With repeated data and considering the probability distribution of the duration of each activity, we can see which activities are most often to blame, which may motivate more detailed exploration of why this is the case. This analysis can also help to identify the most critical activities, so that one can assign more or less resources to a specific job. It can also give us information about how reasonable the overall deadline D is.

We aim to assign the time limits dur_j to be *slack-free* and *balanced*, given the overall time limit D (and potentially other constraints on the individual

* This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289 which is co-funded under the European Regional Development Fund, and by United Technologies Corporation under the Insight-UCC Collaboration Project.

time limits). Slack-free means that it is impossible to increase the limits whilst still satisfying the constraints. If the time limits vector is not slack-free then the durations of activities could exceed these time limits whilst still being consistent with the constraints.

Being balanced is a more complex notion, but the fundamental idea is being fair to the different activities. We introduce parameterised forms of each time limit $\text{dur}_j = f_j(\alpha)$, chosen so that for any given value of α, and for any activities i and j, a time limit of $f_i(\alpha)$ for the duration of activity i is an equally reasonable requirement as a time limit $f_j(\alpha)$ for the duration of activity j. These functions f_j, which we call *commensuracy functions*, are assumed to be continuous and strictly increasing. Given these functions f_j, a collection of parameters $r(j) : j \in \{1, \ldots, n\}$, generates a collection of time limits $\text{dur}_j = f_j(r(j)) : j \in \{1, \ldots, n\}$, so that it is sufficient to choose the vector r of parameters. We sometimes abbreviate $r(j)$ to r_j.

Linear Case: We first consider the simple case where all the activities are in series. In this case, we must have $\sum_{j=1}^{n} \text{dur}_j \leq D$, or in terms of a parameters vector r, $\sum_{j=1}^{n} f_j(r_j) \leq D$. Then r is slack-free if and only if $\sum_{j=1}^{n} f_j(r_j) = D$. Since there is only one complete path, all the n activities are similar in the sense described above, so to also satisfy the basic balance property we need that for all $j = 1, \ldots, n$, $r_j = \alpha$, for the unique value of α such that $\sum_{j=1}^{n} f_j(\alpha) = D$.

For more general networks, the situation is more complicated. It will typically not be possible to set the values of r to be all equal, without breaking the slack-free property. It would mean potentially penalising an activity j whose duration is greater than $f_j(r_j)$, even though the overall time limit D (the makespan limit) is still maintained. However, we do not want to penalise any activities unnecessarily.

Thus, the input of the problem is a graph $\mathcal{G}$, where with each edge j, representing an activity, is associated a commensuracy function f_j, and an overall time limit D. The output is a balanced and slack-free deadline dur_j for each activity j.

In this paper we focus on balancing schedules by using a standard notion of fairness, based on maximising leximin, which is a refinement of maximising the minimum value (see e.g., [24, 13] for a deep investigation of fairness in many different contexts). The final output is fair in the sense that there is no way to increase the parameter r_j of an activity j (and therefore the corresponding time limit $f_j(r_j)$) without decreasing another parameter r_i which is lower or equal.

A standard form of algorithm for obtaining the max leximin element for many problems is sometimes referred to as the *water filling* algorithm[1]; the idea is to increase the levels of each component together until one of the constraints becomes tight; this gives a maximin solution; the components in tight constraints are then fixed (since reducing any such component will give a solution with worse min value, and increasing any such component will give a vector that fails to satisfy the tight constraint). The non-fixed components are increased again until

[1] This is different from the classic water pouring/filling algorithms for allocating power [17].

a new constraint involving one of them becomes tight, and so on. We prove the correctness of this algorithm in a rather general setting, which makes no assumption of convexity of the spaces (in contrast with the unifying framework in [18]).

Different Forms of Commensuracy Function: There are different approaches for generating the commensuracy functions f_j. One kind of method involves making use of a probability distribution over the durations of activity j (or an approximation of this based on past data). For instance, one might define $f_j(\alpha)$ to be equal to $\mu_j + \alpha\sigma_j$, where μ_j is the mean of the distribution of the jth duration, and σ_j is its standard deviation. Alternatives include $f_j(\alpha) = \mu_j + \alpha\sigma_j^2$; or the quantile function: $f_j(\alpha)$ is the value d such that the probability that the duration is less than or equal to d is equal to α.

We first, in Section 2, give an example to illustrate the method and our notation. In Section 3 we define some notation that we use throughout the paper. Section 4 describes the maximum leximin method for balancing the schedules, and defines a simple iterative method that we prove generates the unique most balanced schedule. Other balancing methods are also possible, but may lack some natural properties. Section 5, describes the experimental testing, with the related work being discussed in Section 6, and Section 7 concluding.

2 Running Example

Fig. 1: Activities graph

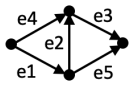


Table 1: parameters

	e_1	e_2	e_3	e_4	e_5
μ_j	5	5	5	4	4
σ_j	1	1	3	2	2

Consider the graph in Figure 1, where each edge e_j is associated with the commensuracy function $f_j(r_j) = \mu_j + r_j\sigma_j$. Assuming a global deadline $D = 20$, and the parameters of the activities shown in Table 1, we want to compute a slack-free and balanced vector r. Starting with $r_j = 0$ for each activity j, we increase the values of all the parameters r_j at the same rate until we find the first complete path π_1 with length D. In this example, given the assignment $r = (1, 1, 1, 1, 1)$, the path $\pi_1 = \{e_1, e_2, e_3\}$ has length $D(= 20)$; we can then fix the values r_j of the activities in π_1 (i.e. $r(1) = r(2) = r(3) = 1$) and keep increasing the remaining non-fixed r_j. Repeating this procedure until all the parameters are fixed, we obtain a slack-free assignment r that is balanced w.r.t. (with respect to) the commensuracy functions f_j. In the current example this requires three iterations: the first to fix the parameters of π_1, the second to fix $r(4) = 4$ ($\pi_2 = \{e_4, e_3\}$), and the third to fix $r(5) = 5$ ($\pi_3 = \{e_1, e_5\}$). The final assignment is therefore $r = (1, 1, 1, 4, 5)$ with associated time limits vector $\text{dur} = (6, 6, 8, 12, 14)$.

3 Formal Definitions

Graphical structure: We assume a finite directed acyclic graph $\mathcal{G}$ containing a source node and a sink node and n edges, each of which we label with a different value in $\{1, \ldots, n\}$. Apart from the source (with only out-edges) and sink (with only in-edges), every node has at least one in-edge and at least one out-edge. A *complete path* is defined to be a path from source to sink; we identify this with its set of edges π. The assumptions above imply that every edge is in at least one complete path. We write the set of complete paths as $\mathcal{C}_\mathcal{G}$. As discussed in Section 1, each edge is intended to represent an activity required for making a product, with the topology encoding precedence constraints.

Commensuracy Functions f_j: For each $j \in \{1, \ldots, n\}$, we assume a strictly monotonic continuous function f_j from closed interval I to the non-negative reals. We write $I = [L_I, U_I]$, where we make some assumptions on I below.

Assignments: A *complete assignment* is a function from $\{1, \ldots, n\}$ to I. The set of all complete assignments is written as CA. Given a complete assignment r, we consider $f_j(r_j)$ as the length of edge j in the graph $\mathcal{G}$, which we also consider as the time limit (maximum duration) of the jth activity.

An *assignment* is a function b from some subset B of $\{1, \ldots, n\}$ to I. We write $Dom(b) = B$, and write AS as the set of all assignments. A *partial assignment* is an element of PA = AS $\setminus$ CA, i.e., a function from a proper subset of $\{1, \ldots, n\}$ to I. For convenience we define $\diamondsuit$ as the empty assignment, i.e., the (trivial) function from $\emptyset$ to I.

Consider two assignments $a : A \to I$ and $b : B \to I$, where $A \subseteq B$. We say that a is the projection of b to A if for all $j \in A$, $a(j) = b(j)$. We write $a = b^{\downarrow A}$. We also then say that b *extends* a. If, in addition, $a \neq b$ then b *strictly extends* a. For instance (see the running example in Section 2), complete assignment $(1, 1, 1, 4, 5)$ strictly extends partial assignment $(1, 1, 1, _, _)$.

Pareto Dominance: Consider two complete assignments r and s. We write $r \geqq s$ if and only if for all $j \in \{1, \ldots, n\}$, $r(j) \geq s(j)$. We say that r *Pareto-dominates* s if $r \geqq s$ and $r \neq s$, and write $r \gneqq s$. We say that r is Pareto-undominated in a set $T \subseteq$ CA if $r \in T$ and there exists no element of T that Pareto-dominates r.

Length of Path: With the graphical case of $\mathcal{C}_\mathcal{G}$, for assignment s and $\pi \in \mathcal{C}_\mathcal{G}$ such that $\pi \subseteq Dom(s)$, we define $Len_s(\pi) = \sum_{j \in \pi} f_j(s_j)$. This is the length of π, given s. We also define $Makespan(r) = \max_{\pi \in \mathcal{C}_\mathcal{G}} Len_r(\pi)$, which is the length of the longest complete path, given complete assignment r.

Consistent and Slack-Free Assignments

We will consider a somewhat more general setting than that purely based on constraints on the maximum lengths of paths, enabling our approach to be more generally applicable. We can add, for instance, upper bounds on time limits and on the lengths of incomplete paths, representing e.g., completion of a part of the product (implemented using a modified makespan relative to an internal

node in the graph) as well as more complex constraints, which may cause the set of consistent complete assignments (and also the set of feasible duration vectors) to be non-convex. This can include upper bounds on more complex sums and averages, such as OWAs (ordered weighted averages) [21, 7], and use of soft minimums to represent such constraints as *either Part-1 is completed early or Part-2 is*. (In addition, direct constraints on the complete assignments in the preference space may well lead to a non-convex space of feasible duration vectors, via non-linear commensuracy functions.)

We are especially interested in an upper bound constraint on the makespan, leading to a constraint for each complete path π that can be written as $Len_r(\pi) - D \leq 0$. The left-hand-side is a continuous function of r that is strictly increasing in each component of π. We consider more general constraints of this form.

We assume a set $\mathcal{C}$ of non-empty subsets of $\{1, \ldots, n\}$, such that every $j \in \{1, \ldots, n\}$ is in some element of $\mathcal{C}$, and associated with each $\pi \in \mathcal{C}$ is a continuous function H_π that maps assignments with domain π to real numbers, and that is strictly increasing w.r.t. each argument in π. We use H_π to express constraints, for example, a time limit on a sub-path. For complete assignment r we also write $H_\pi(r)$ as an abbreviation for $H_\pi(r^{\downarrow\pi})$.

We also have a value L_j associated with each $j \in \{1, \ldots, n\}$ (the *lower bound* for component j), and we assume, without loss of generality, that $L_j \geq L_I$.

Let $r \in \mathrm{CA}$ be a complete assignment and let $\pi \in \mathcal{C}$. We say that r *satisfies the lower bound constraints* if for all $j \in \{1, \ldots, n\}$, $r(j) \geq L_j$. We say that r *satisfies [the constraint for]* π if $H_\pi(r) \leq 0$.

We also say that π is *tight w.r.t.* r if $H_\pi(r) = 0$. We define $UT(r)$ to be the union of all $\pi \in \mathcal{C}$ such that π is tight w.r.t. r.

In the running example, for each complete path π, $H_\pi(r) = Len_r(\pi) - D \leq 0$ is the constraint representing the upper bound limit D for the sum of the durations of the activities in the path π under the assignment r. $H_{\pi_3}(1, 1, 1, 4, 4) = H_{\pi_3}(1, _, _, _, 4) = f_1(1) + f_5(4) - 20 = -2$, with $\pi_3 = \{e_1, e_5\}$. Thus, π_3 is not tight w.r.t. $(1, 1, 1, 4, 4)$. We have $UT(1, 1, 1, 4, 4) = \{e_1, e_2, e_3, e_4\}$.

Defining consistency, $\mathcal{R}$ and $\mathcal{S}$: We say that complete assignment r is *consistent* if r satisfies the lower bound constraints and each $\pi \in \mathcal{C}$. We write $\mathcal{S}$ for the set of consistent complete assignments. Partial assignment b is said to be *consistent* if there exists an element of $\mathcal{S}$ extending b. We say that consistent complete assignment r is *slack-free* if for all $j \in \{1, \ldots, n\}$ there exists some $\pi \in \mathcal{C}$ containing j that is tight with respect to r; in other words, if $UT(r) = \{1, \ldots, n\}$. It can be shown that r in $\mathcal{S}$ is slack-free if and only if r is Pareto-undominated in $\mathcal{S}$. We write $\mathcal{R}$ for the set of slack-free consistent complete assignments.

Assumptions on L_I, U_I and on the lower bounds: We assume that the empty assignment $\diamondsuit$ is consistent, i.e., that $\mathcal{S}$ is non-empty. Because of the monotonicity of each H_π, this is equivalent to the assumption that $r^L \in \mathcal{S}$, where, for all $j \in \{1, \ldots, n\}$, $r^L(j)$ is defined to be L_j. We also assume that every partial assignment can be extended to a complete assignment not in $\mathcal{S}$. This is equivalent to the assumption that for each $i \in \{1, \ldots, n\}$, $r^i_L \notin \mathcal{S}$, for r^i_L defined by $r^i_L(i) = U_I$ and $r^i_L(j) = L_j$ for $j \in \{1, \ldots, n\} \setminus \{i\}$.

The Case of $\mathcal{C}_\mathcal{G}$: When the set of constraints is $H_\pi(r) = Len_r(\pi) - D \leq 0$ for each complete path $\pi \in \mathcal{C}_\mathcal{G}$, then, the definition of $\mathcal{S}$ simplifies to: $r \in \mathcal{S} \iff r$ satisfies the lower bound constraints and $Makespan(r) \leq D$. This helps computationally since it enables us to deal with the exponential number of constraints in a compact way. For this case, we can show a further characterisation[2] of the set $\mathcal{R}$ of consistent slack-free complete assignments: for complete assignment r satisfying the lower bound constraints, $r \in \mathcal{R}$ if and only if every element of $\mathcal{C}_\mathcal{G}$ is tight w.r.t. r.

Proposition 1. *Suppose $\pi \in \mathcal{C}_\mathcal{G}$ is a complete path and $\pi \subseteq UT(r)$ for some complete assignment $r \in \mathcal{S}$. Then, π is tight w.r.t. r, i.e., $Len_r(\pi) = D$. For $r \in \mathcal{S}$, we have $r \in \mathcal{R}$ if and only if every element of $\mathcal{C}_\mathcal{G}$ is tight w.r.t. r.*

4 Leximin Maximising: Iterative Method

We aim to find a most balanced consistent slack-free complete assignment r. For the process graph $\mathcal{G}$, we then can define the time limit of activity j to be $f_j(r_j)$. The basic idea is to maximise the minimum value (over all the n co-ordinates of r). However, there are many such vectors; it is thus natural to iterate this process, which leads to leximin maximising.

4.1 Most Balanced Schedule

Given complete assignment r, define $r^\uparrow$ to be the vector in $\mathbb{R}^n$ formed by permuting the co-ordinates of r in such a way that $r^\uparrow(1) \leq r^\uparrow(2) \leq \cdots \leq r^\uparrow(n)$.

We define the leximin order relation $\leq_{lexm}$ by $s \leq_{lexm} r$ if and only if either $s^\uparrow = r^\uparrow$, or there exists some $i \in \{1, \ldots, n\}$ such that $s^\uparrow(i) < r^\uparrow(i)$, and for all $j < i$, $s^\uparrow(j) = r^\uparrow(j)$. It follows easily that $\leq_{lexm}$ is a total pre-order (a transitive and complete relation); also, if r is a complete assignment, and r' is generated from r by permuting the n co-ordinates in some way, then r and r' are equivalent in the order. We always have $r \geqq s \Rightarrow r \geq_{lexm} s$. The maximal leximin r in $\mathcal{S}$ are all those vectors $r \in \mathcal{S}$ such that for all $s \in \mathcal{S}$, $r \geq_{lexm} s$.

The main result of this section, Theorem 1, implies that there exists a unique leximin-maximal element in $\mathcal{R}$ (and also in $\mathcal{S}$), and this can be obtained through a sequence of maximisations over one-dimensional sets. This allows efficient implementation, as discussed in Section 5.

4.2 The Basic Iteration Operation

We will define an operation that takes a consistent partial assignment b and generates an assignment b_* that strictly extends b. Iterating this operation will

[2] For space reasons, almost all the proofs have been omitted; they can be found in the longer version [20], which also contains auxiliary results and many more details about the implementation and experimental testing.

lead to a complete assignment in $\mathcal{S}$, which we prove later (in Theorem 1) to be the unique maximum leximin complete assignment extending b.

Notation b^α, $\widetilde{b}$, Z_b, $\gamma(b)$, Fix(b): We will define a method for extending a consistent partial assignment b to a complete assignment $\widetilde{b}$ in $\mathcal{S}$. For partial assignment $b \in$ PA and $\alpha \in I$ we first define b^α to be the complete assignment extending b given by, for $j \in \{1, \ldots, n\} \setminus Dom(b)$, $b^\alpha(j) = \max(\alpha, L_j)$. For partial assignment $b \in$ PA we define Z_b to be the set of all $\alpha \in I$ such that $b^\alpha \in \mathcal{S}$. This is non-empty if and only if b is consistent. Our assumption about U_I implies that $U_I \notin Z_b$.

For consistent partial assignment b, we define $\gamma(b)$ to be the supremum of Z_b. We also define $\widetilde{b}$ to be $b^{\gamma(b)}$. We write $UT(\widetilde{b}\,)$ as Fix(b); these are the variables that are in some π that is tight w.r.t. $\widetilde{b}$ (and they are fixed given b at the end of each stage of the iterative sequence described in Section 4.3).

The lemma below, giving some basic properties, can be proved making use of the fact that $H_\pi(b^\alpha)$ is an increasing continuous function of α.

Lemma 1. *Suppose that $b \in$ PA is a consistent partial assignment. Then, $\gamma(b) \in Z_b$ and $\widetilde{b} \in \mathcal{S}$. Also, there exists $\pi \in \mathcal{C}$ such that $\pi \not\subseteq Dom(b)$ and π is tight w.r.t. $\widetilde{b}$. Thus, Fix$(b) \not\subseteq Dom(b)$.*

Definition of b_*: For consistent partial assignment $b \in$ PA, we define b_* to be the projection of $\widetilde{b}$ to $Dom(b) \cup$ Fix(b). Thus, $Dom(b_*) = Dom(b) \cup$ Fix(b).

In the running example with $I = [0, 10]$ and $b_2 = (1, 1, 1, _, _)$, we have $Dom(b_2) = \{e_1, e_2, e_3\}$, $b_2^\alpha = (1, 1, 1, \alpha, \alpha)$, $\gamma(b_2) = 4$, $\widetilde{b_2} = (1, 1, 1, 4, 4)$, Fix$(b_2) = \{e_1, e_2, e_3, e_4\}$, and $(b_2)_* = (1, 1, 1, 4, _)$.

Proposition 2. *Assume that partial assignment b is consistent. Then b_* is consistent and strictly extends b. If $Dom(b_*) \neq \{1, \ldots, n\}$ then $\widetilde{b_*} \gneqq \widetilde{b}$ and $Fix(b_*) \supsetneqq Fix(b)$.*

The following proposition gives key properties related to leximin dominance. Regarding (i), the point is that for all $j \in$ Fix(b) there exists a $\pi \in \mathcal{C}$ that is tight w.r.t. $\widetilde{b}$, and also with respect to any extension of b_*. Strict monotonicity of H_π implies that we cannot increase the value of any such j from its value in b_* without violating the constraint π. Thus, r is equal to $\widetilde{b}$ on Fix(b), and so, r extends b_* (since it extends b).

(ii) includes an interesting (very partial) converse of the property $r \geqq s \Rightarrow r \geq_{lexm} s$. The rough idea is that if $r \in \mathcal{S}$ and r extends b and $r \not\geqq b^\alpha$ then there exists $j \in \{1, \ldots, n\} \setminus Dom(b)$ such that $r(j) < \alpha$, which leads to $b^\alpha >_{lexm} r$. Then, using $\alpha = \gamma(b)$ and so, $b^\alpha = \widetilde{b}$, we can chain (ii) and (i) to obtain (iii).

Proposition 3. *Let r be an element of $\mathcal{S}$ that extends assignment b.*

(i) If for all $j \in Dom(b_*)$, $r(j) \geq b_*(j)$ then r extends b_*.
(ii) For any α such that $b^\alpha \in \mathcal{S}$, $r \geq_{lexm} b^\alpha \iff r \geqq b^\alpha$.
(iii) If $r \geq_{lexm} \widetilde{b}$ then r extends b_*.

4.3 The Iterative Sequence Generated from b

Given consistent partial assignment b, we define a sequence of assignments, $b_1, b_2, \ldots, b_m$, in an iterative fashion, as follows:

Define $b_1 = b$. Assume now that b_i has been defined, for some $i \geq 1$. If b_i is consistent and $Dom(b_i) \neq \{1, \ldots, n\}$, we let b_{i+1} equal $(b_i)_*$; otherwise, we end the sequence with i, and let $m = i$. We call $b_1, b_2, \ldots, b_m$ the *iterative sequence of assignments generated from* b_1, and we say that b_m is its *result*.

We are mainly interested in the case in which the initial partial assignment b_1 is the empty assignment $\diamondsuit$. However, allowing other b_1 enables a simple representation of a situation in which certain of the components are fixed in advance, i.e., some of the durations are fixed. Table 2 shows the iterative sequence of assignments generated from $b_1 = \diamondsuit$ in the running example, where $b_m = b_4 = (b_3)_*$.

Table 2: Progress of the algorithm

i	$\gamma(b_i)$	$\widetilde{b_i}$	$(b_i)_*$
1	1	$(1,1,1,1,1)$	$(1,1,1,_,_)$
2	4	$(1,1,1,4,4)$	$(1,1,1,4,_)$
3	5	$(1,1,1,4,5)$	$(1,1,1,4,5)$

Proposition 2 implies that each b_i in the sequence is consistent, and strictly extends the previous element. This implies that the sequence terminates with some complete assignment $t = b_m$, with each earlier b_i being a partial assignment. The other parts of Proposition 2 can be used to show that $t \geqq \widetilde{b_i}$ and $UT(t) \cup Dom(b_1) = \{1, \ldots, n\}$.

Proposition 4. *Consider the iterative sequence of assignments $b_1, b_2, \ldots, b_m$, generated by consistent assignment b_1, and let $t = b_m$ be its result. Then, t is a complete assignment in $\mathcal{S}$ that extends each b_i, and for all $i = 1, \ldots, m-1$, $t \geqq \widetilde{b_i}$, and b_i is a consistent partial assignment. Also, $UT(t) \cup Dom(b_1) = \{1, \ldots, n\}$.*

Propositions 3 and 4 lead to the following theorem, which shows that there is a uniquely maximally leximin element in $\mathcal{S}$ (and in $\mathcal{R}$), and the iterative sequence can be used as the basis of an algorithmic procedure for finding it.

Theorem 1. *The result of the iterative sequence of assignments generated from b is the unique maximal leximin element in $\mathcal{S}_b$, where $\mathcal{S}_b$ is the set of elements of $\mathcal{S}$ that extend b. If b is the empty assignment then the result is in $\mathcal{R}$ and is thus the unique maximal leximin element in $\mathcal{S}$ and the unique maximal leximin element in $\mathcal{R}$.*

Proof: Let $b_1, b_2, \ldots, b_m$ be the iterative sequence of assignments generated by $b = b_1$ and with result $t = b_m$. By Proposition 4, t is in $\mathcal{S}_b$, and for all $i = 1, \ldots, m-1$, $t \geqq \widetilde{b_i}$ and b_i is a consistent partial assignment.

Consider any element r of $\mathcal{S}_b$ such that $r \geq_{lexm} t$. This implies that, for all $i = 1, \ldots, m-1$, $r \geq_{lexm} \widetilde{b_i}$, because $t \geqq \widetilde{b_i}$ (and thus, $t \geq_{lexm} \widetilde{b_i}$). Since

r extends $b = b_1$, Proposition 3(iii) applied iteratively shows that r extends each b_i, for $i = 1, \ldots, m$, and, in particular, r extends t. Since, t is a complete assignment, we have $r = t$.

We have shown that for any $r \in \mathcal{S}_b$ if $r \geq_{lexm} t$ then $r = t$. Thus, if $r \neq t$ then $r \not\geq_{lexm} t$, i.e., $t >_{lexm} r$. Thus, t is the unique maximally leximin element in $\mathcal{S}_b$.

Now, consider the case when b is the empty assignment $\diamondsuit$. Since $\mathcal{S}_\diamondsuit = \mathcal{S}$, assignment t is then the unique maximal leximin element in $\mathcal{S}$. Proposition 4 implies that $UT(t) = \{1, \ldots, n\}$, and so $r \in \mathcal{R}$. Therefore, because $\mathcal{R} \subseteq \mathcal{S}$, assignment t is the unique maximal leximin element in $\mathcal{R}$. □

A vector t is said to be *max-min fair on* $\mathcal{X} \subseteq \mathbb{R}^n$ if and only if for all $s \in \mathcal{X}$ and $j \in \{1, \ldots, n\}$ if $s(j) > t(j)$ then there exists $i \in \{1, \ldots, n\}$ such that $s(i) < t(i) \leq t(j)$. Thus, increasing some component $t(j)$ must be at the expense of decreasing some already smaller component $t(i)$. If there exists a max-min fair element, then it is unique and equals the unique maximum leximin element [18]. Theorem 1 can be used to show that there is a max-min fair element in $\mathcal{S}$, i.e., the max leximin element t. The idea behind the proof is that if max-min fairness were to fail for t, then one can show that there would exist s and j such that $s(j) > t(j)$ and for all $i \in \mathit{Dom}(b_k)$, $s(i) \geq t(i)$, where k is minimal in the iterative sequence such that $\mathit{Dom}(b_k) \ni j$. Applying Proposition 3(i) iteratively would imply that s extends b_k, contradicting $s(j) > t(j) = b_k(j)$.

Corollary 1. *For any consistent partial assignment b, the maximum leximin element of $\mathcal{S}_b$ is max-min fair on $\mathcal{S}_b$.*

Corollary 2. *For the maximum leximin element r in $\mathcal{S}$, every component is maximal w.r.t. some constraint, i.e., for each $j \in \{1, \ldots, n\}$ there exists $\pi \in \mathcal{C}$ such that j is maximal w.r.t. π. For the graph case when $\mathcal{C} = \mathcal{C}_\mathcal{G}$, for r in $\mathcal{S}$, we have r is the maximum leximin element in $\mathcal{S}$ if and only if r is slack-free and every component is maximal w.r.t. some constraint.*

5 Implementation

We have implemented a version of the water filling algorithm for the graph-based case using $\mathcal{C}_\mathcal{G}$, and with both linear and non-linear commensuracy functions f_j (see Section 3). Our algorithm constructs the iterative sequence generated from the empty assignment $\diamondsuit$ (see Section 4.3). To implement this, we need, for partial assignment b, to compute b_* (see Section 4.2), by first computing $\gamma(b)$, and setting b_* to be the projection of $b^{\gamma(b)}$ to $\mathrm{Fix}(b)$, where $\mathrm{Fix}(b)$ is determined using a simple forward and backward dynamic programming algorithm. We use an obvious binary/logarithmic search algorithm to approximate $\gamma(b)$ within a chosen number $\epsilon > 0$, using the fact that $\gamma(b)$ is the maximal real β such that $\mathit{Makespan}(b^\beta) \leq D$. We also implemented a variation of this iterative approach for computing $\gamma(b)$, based on iterating over paths: given an upper bound β for $\gamma(b)$, we generate the longest path π w.r.t. b^β and then update β to a better

upper bound β', defined to be the unique solution of the equation $Len_{b^{\beta'}}(\pi) = D$; we iterate until $Makespan(b^{\beta}) = D$. In our experimental testing we were able to solve problems with hundreds of activities in few seconds [20].

6 Related Work

A branch of research, that is somewhat related to our problem of computing time limits within a network of activities, is that concerned with the minimization of the *tardiness* of a set of jobs [22, 4, 9, 3]. The tardiness is defined as $max(C_i - d_i, 0)$, where C_i is the completion time for a job and d_i is the due time. In our scenario, dur_i is equivalent to d_i, and we want to evaluate the *tardiness* as well; but in this case d_i is given as input, and the goal is finding the best scheduling of jobs in an assembly line where a machine is able to treat only one job at a time. Flexible constraints for scheduling (under uncertainty) have also been considered in [5].

Max leximin (and max-min) fairness has been widely studied and applied. For example, there are applications for balancing social welfare in markets [8], for a price-based resource allocation scheme [12], for kidney exchange [23], and for allocating unused classrooms [11]. Optimising leximin on constraint networks is studied in [2], and for systems of fuzzy constraints in [6]. Regarding the work which, like our framework, uses continuous variables, there is a substantial literature related to bandwidth allocation problems, e.g., [18, 10, 19, 1, 15, 14]; see [16] for a survey of fair optimisation for networks. The most general framework of this kind seems to be that in [18], showing, under relatively general conditions, that the water filling algorithm generates the unique max leximin, which equals the max-min fair solution. Although the applications they focus on are very different from our form of scheduling problem, their theoretical framework and results still apply if the constraints on durations are linear inequalities, since their framework assumes convexity (and compactness) of the space of durations. In contrast, our framework makes no assumption of convexity.

7 Summary and Discussion

We have explored the problem of fairly assigning time limits for processes in manufacturing a product whose duration can be uncertain, subject to a deadline. We proved that a simple iterative procedure (a version of the water filling algorithm) can be used to generate the unique most balanced solution, w.r.t. max leximin, in a very general setting, not making any assumptions of convexity. This allows a wide range of side constraints to be added to the problem, whilst maintaining the same structure of the algorithm. We go on to prove, in this very general setting, that the maximum leximin element is still max-min fair, and discuss further properties. The experimental results of our implementation indicate that it is scalable to large problems. In the future, we plan to apply the method for real industrial problems, and to develop automatic methods for suggesting remedial actions for problematic schedules.

References

1. Bonald, T., Massoulié, L., Proutiere, A., Virtamo, J.: A queueing analysis of max-min fairness, proportional fairness and balanced fairness. Queueing systems **53**(1-2), 65–84 (2006)
2. Bouveret, S., Lemaître, M.: Computing leximin-optimal solutions in constraint networks. Artificial Intelligence **173**(2), 343–364 (2009)
3. Chen, J.F.: Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. The International Journal of Advanced Manufacturing Technology **29**(5), 557–563 (2006)
4. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is np-hard. Mathematics of operations research **15**(3), 483–495 (1990)
5. Dubois, D., Fargier, H., Fortemps, P.: Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. European Journal of Operational Research **147**(2), 231–252 (2003)
6. Dubois, D., Fortemps, P.: Computing improved optimal solutions to max-min flexible constraint satisfaction problems. European Journal of Operational Research **118**(1), 95–126 (1999)
7. Fodor, J., Marichal, J.L., Roubens, M.: Characterization of the ordered weighted averaging operators. IEEE Transactions on Fuzzy Systems **3**(2), 236–240 (1995)
8. Gopinathan, A., Li, Z.: Strategyproof auctions for balancing social welfare and fairness in secondary spectrum markets. In: INFOCOM, 2011 Proceedings IEEE. pp. 3020–3028. IEEE (2011)
9. Ho, J.C., Chang, Y.L.: Heuristics for minimizing mean tardiness for m parallel machines. Naval Research Logistics (NRL) **38**(3), 367–381 (1991)
10. Huang, X.L., Bensaou, B.: On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation. In: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing. pp. 221–231. ACM (2001)
11. Kurokawa, D., Procaccia, A.D., Shah, N.: Leximin allocations in the real world. In: Roughgarden, T., Feldman, M., Schwarz, M. (eds.) Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015. pp. 345–362. ACM (2015)
12. Marbach, P.: Priority service and max-min fairness. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. vol. 1, pp. 266–275. IEEE (2002)
13. Moulin, H.: Fair division and collective welfare. MIT press (2004)
14. Nace, D., Orlin, J.B.: Lexicographically minimum and maximum load linear programming problems. Operations Research **55**(1), 182–187 (2007)
15. Nace, D., Pioro, M., Doan, L.: A tutorial on max-min fairness and its applications to routing, load-balancing and network design. In: 4th IEEE International Conference on Computer Sciences Research, Innovation and Vision for the Future (RIVF 2006) (2006)
16. Ogryczak, W., Luss, H., Pióro, M., Nace, D., Tomaszewski, A.: Fair optimization and networks: A survey. Journal of Applied Mathematics **2014** (2014)
17. Palomar, D.P., Fonollosa, J.R.: Practical algorithms for a family of waterfilling solutions. IEEE transactions on Signal Processing **53**(2), 686–695 (2005)
18. Radunović, B., Boudec, J.Y.L.: A unified framework for max-min and min-max fairness with applications. IEEE/ACM Transactions on Networking (TON) **15**(5), 1073–1083 (2007)

19. Tassiulas, L., Sarkar, S.: Maxmin fair scheduling in wireless networks. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. vol. 2, pp. 763–772. IEEE (2002)
20. Toffano, F., Wilson, N.: Balancing Schedules Using Maximum Leximin (Extended version including proofs). Unpublished Document (2019)
21. Yager, R.R.: On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Trans. Syst. Man Cybern. **18**(1), 183–190 (Jan 1988). https://doi.org/10.1109/21.87068
22. Yalaoui, F., Chu, C.: Parallel machine scheduling to minimize total tardiness. International Journal of Production Economics **76**(3), 265–279 (2002)
23. Yilmaz, Ö.: Kidney exchange: An egalitarian mechanism. J. Economic Theory **146**(2), 592–618 (2011)
24. Young, H.P.: Equity: in theory and practice. Princeton University Press (1995)