URL: http://www.elsevier.nl/locate/entcs/volume40.html 24 pages

The complexity of scheduling TV commercials

Klemens Hägele [1]

IBM UBG, Kennedypark, Kaiserswertherstr. 117,
40476 Düsseldorf, Germany

Colm Ó Dúnlaing [2]

Mathematics, Trinity College,
Dublin 2, Ireland

Søren Riis [3]

Queen Mary, University of London,
United Kingdom

Abstract

Television commercial scheduling is a generalised form of partition problem, but the lengths involved are rather small, leaving the complexity of the problem unclear. This paper shows that the related problem of colour-restricted spot scheduling is **NP**-complete, even when the break-lengths are bounded (at most 18 units). We also show that scheduling unit-length spots is easy.

The paper is intended to be self-contained.

1 Introduction

The main source of income for a commercial TV station is advertising. The broadcasting day is interspersed with advertising 'breaks' typically 3 minutes long. In the business, advertisements are called 'spots.'

Typical spot-lengths are 7 seconds, 15, 22, 30, 45, 60, 90, and 120.

It is a rule of television advertising that competing products should not be advertised within the same break. Hence products are separated into 'clash

[1] Email: Klemens.Haegele@de.ibm.com
[2] Email: odunlain@maths.tcd.ie
[3] Email: smriis@dcs.qmw.ac.uk

groups,' and products within the same clash group should not be advertised in the same break.[4]

Thus the scheduling problem involves advertising *breaks*, *spots* to be broadcast, and spot *colours.* The colour of a spot indicates the kind of product being advertised, so spots of the same colour cannot be broadcast in the same break.

Around 1970, Brown [i,ii] investigated this problem, but we know of no other research into the problem. It is certainly of interest in the context of **NP**-completeness.

This paper presents our results about the complexity of spot scheduling. The presentation does not assume familiarity with **NP**-completeness, and it may serve as an introduction to the non-specialist.

Some of these results were presented at MFCSIT2000, the first Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, at University College, Cork, in July 2000.

2 NP completeness

(2.1) Polynomial time. Computational complexity is often concerned with polynomial-time computability. A computation problem has a polynomial-time solution if there is a polynomial $f(n)$ such that when the input has size n (suitably measured, as in 4.4), the time to compute the result is bounded by $f(n)$.

Definition 2.2 *A computational task is 'easy' if it can be done in polynomial time. Otherwise it is 'difficult.'*

(2.3) Example: linear equations. For example, there are several ways to solve a system of n linear equations in n unknowns. One is Cramer's Rule, which, if applied blindly, will compute the answer with something like $n \times n!$ operations. More than 39 million operations would be needed when $n = 10$. On the other hand, Gaussian elimination would produce the answer in about n^3 operations. When $n = 10$, about 1000 operations would be needed. Gaussian elimination is polynomial time, Cramer's Rule certainly isn't.

(2.4) Example: linear programming. Linear programming means, roughly speaking, finding solutions to a system of linear inequalities

$$AX \leq B$$

where A is a matrix and B a column-vector. Dantzig's Simplex Method of 1947 [iii] is very effective, but known to take exponential time on certain problems. Khachian's algorithm of the late 1970s runs in polynomial time but is useless in practice. Karmakar's algorithm of 1984 [vi] is another polynomial-time

[4] This is a simplification of the real situation. Strictly speaking, the 'clash' relationship is not transitive. Sometimes non-competing products clash because their brand-names are similar.

algorithm which is much more effective in practice. The point, however, is that linear programming is solvable in polynomial time.

On the other hand, integer programming — linear programming where the solution vectors are required to have integer components — is a harmless-looking variant which is *not* known to have a polynomial-time solution. It is **NP**-complete (2.11): strong evidence that no efficient method exists.

(2.5) Definition of NP. The class **NP** is, strictly speaking, the class of languages which can be accepted in polynomial time on a nondeterministic Turing machine. See [v] for a detailed survey of the subject.

Here is an equivalent, and simple, definition of **NP**.

Definition 2.6 *The class* **NP** *consists of those problems whose solutions are easy to verify.*

(2.7) Example: satisfiability of Boolean expressions. Another example of an **NP** problem is, given a Boolean expression Φ, whether some truth assignment will satisfy Φ, i.e., make it true. Obviously it is easy to certify that a given truth-assignment satisfies Φ. Therefore the satisfiability problem is in **NP**. But is it easy to *decide* whether such a truth-assignment exists? At present, it is not known one way or the other, but no efficient, general, method has yet been discovered for solving such problems, and it is believed that none exists.

The problem is **NP**-complete (2.11,3.3). The satisfiability problem is important and will be discussed further in the next section.

Definition 2.8 *Given two computational problems X and Y, an* easy reduction *of X to Y is an easily-computable function g such that for every instance x of problem X, $g(x)$ is an instance of problem Y, and x has a solution if and only if $g(x)$ has a solution.*

Definition 2.9 *Given two computational problems X and Y, write $X \leq Y$ if X can be easily reduced to Y (i.e., there exists an easy reduction of X to Y).*

For example, the satisfiability problem is easily reduced (2.8) to integer programming.

Corollary 2.10 *If X is difficult (see 2.2) and $X \leq Y$ then Y is difficult. (Proof straightforward.)* $\square$

Definition 2.11 *A problem Y is* **NP**-complete *if* (i) $Y \in$ **NP** *and* (ii) *for* every *X in* **NP***, $X \leq Y$.*

So if **NP** contains any difficult problem — one with no polynomial-time solution — then all **NP**-complete problems are difficult.

Corollary 2.12 *If X is an* **NP***-complete problem, $Y \in$* **NP***, and $X \leq Y$, then Y is* **NP***-complete. (Proof straightforward.)* $\square$

(2.13) The class **P** is the class of languages which can be accepted in polynomial time on a (deterministic) Turing machine. More simply,

Definition 2.14 *The class* **P** *consists of those problems whose solutions are both easy to find and easy to verify.*

Since 1971, whether or not $\mathbf{P} = \mathbf{NP}$ has been an open problem. If the classes are equal, then all problems in **NP** are easy, including many important problems in Operations Research. If the classes are different, then all **NP**-complete problems are difficult, a disappointing possibility which appears to be true. The general belief is that $\mathbf{P} \neq \mathbf{NP}$.

Among the problems considered in this section, solving linear equations and linear programs are in **P**, while satisfiability of Boolean expressions and integer programming are **NP**-complete.

3 CNF formulae and Cook's Theorem

(3.1) CNF formulae. A *Boolean formula in conjunctive normal form,* or *CNF* for short, is an expression Φ involving *Boolean variables, literals,* and *clauses,* such as

$$(A \vee B \vee C) \ \& \ (A \vee B \vee \overline{C}) \ \& \ (A \vee \overline{B}).$$

- A literal is either a Boolean variable X or its complement $\overline{X}$
- A clause is a disjunction $(L_1 \vee \cdots \vee L_r)$ of literals, such as $(A \vee \overline{B} \vee D)$.
- A CNF is a conjunction $C_1 \ \& \ \cdots \ \& \ C_s$ of clauses.

(3.2) A truth-assignment is a map θ from the Boolean variables to Boolean values 0 and 1, where 0 means **'false'** and 1 means **'true.'** It can be extended to all CNFs involving these variables, according to the following simple rules.

- $\theta(\overline{X}) = 1 - \theta(X)$
- $\theta(L_1 \vee \cdots \vee L_r) = 1$ iff for at least one L_i, $\theta(L_i) = 1$
- $\theta(C_1 \ \& \ \cdots \ \& \ C_s) = 1$ iff for *all* C_j, $\theta(C_j) = 1$.

For example, given

$$\Phi = (A \vee B \vee C) \ \& \ (A \vee B \vee \overline{C}) \ \& \ (A \vee \overline{B}); \quad \theta : \ A \mapsto 0, \ B \mapsto 0, \ C \mapsto 1$$

then

$$\begin{array}{rcl} \theta(\Phi) & = & (0 \vee 0 \vee 1) \ \& \ (0 \vee 0 \vee \overline{1}) \ \& \ (0 \vee \overline{0}) \quad = \\ (0 \vee 0 \vee 1) \ \& \ (0 \vee 0 \vee 0) \ \& \ (0 \vee 1) & = 1 \ \& \ 0 \ \& \ 1 & = \quad 0 \end{array}$$

The assignment θ *satisfies* the formula Φ if $\theta(\Phi) = 1$. The above assignment does not satisfy the given formula, though the formula is satisfiable. The following theorem is the basis of all **NP**-completeness results.

Proposition 3.3 (Cook's Theorem) *[iv,v]. Satisfiability of CNF formulae is an* **NP***-complete problem.* □

It is possible to strengthen Cook's Theorem as follows.

Corollary 3.4 *3SAT, satisfiability of CNFs all of whose clauses have 3 literals, is* **NP***-complete. (See [v].)* □

4 Spot scheduling

The spot-scheduling problem has been mentioned in the Introduction. This section describes the problem formally.

Definition 4.1 *A spot-scheduling problem would be presented as follows*

- *A list of n* spots, *without loss of generality the integers $\{1, \cdots, n\}$.*
- *A list of m* breaks, *without loss of generality the integers $\{1, \cdots, m\}$.*
- *Every spot j has an associated* length *$\ell(j)$ (a positive integer), and* colour *$c(j)$.*
- *Every break i has an associated* size *(or length) $s(i)$ (a positive integer).*

It is required to produce a *schedule* defined as follows.

Definition 4.2 *Given an instance of a spot-scheduling problem, a* valid schedule *S is a map from spots to breaks satisfying*

- **Colour constraint:** *for any break i and distinct spots j and k in $S^{-1}(i)$, $c(j) \neq c(k)$.*
  Equivalently: if $S(j) = S(k)$ then $c(j) \neq c(k)$.
- **Length constraint:** *for any break i,*
  $\sum_{j \in S^{-1}(i)} \ell(j) \quad \leq \quad s(i).$

These constraints are based on commercial practice as discussed in the Introduction.

A well-known **NP**-complete problem is bin-packing, which is the same as spot-scheduling with the colour requirement dropped. We consider the following kind of problem.

(4.3) Multi-partition.[5] An instance of a multi-partition problem consists of a list of n positive integers ℓ_j and m positive integers B_i. The problem is to determine whether there exists a partition of $\{1 \ldots n\}$ into m disjoint sublists, where for the i-th sublist L_i,

$$\sum_{j \in L_i} \ell_j = B_i.$$

This closely resembles a spot-scheduling problem, in that the i-th sublist L_i could correspond to spots scheduled in the i-th break.

Partition and 3-partition. When $m = 2$ we have the so-called partition problem. The 3-partition problem discussed in [v, section 4.2] is a restricted version where $n = 3m$ and each sublist L_i contains 3 elements.

[5] The term is ours, and may not occur elsewhere, but it conveniently covers the problems of interest here.

Definition 4.4 *Let X be a computation problem, and n a suitable measure of input size*[6] *for X. A feature I of X — some integer quantity associated with instances of X — is* 'small' *or* 'short' *if there exists a polynomial $p(n)$ bounding I in absolute value.*

Proposition 4.5 *(See [v].)* (i) *The partition problem is* **NP***-complete, even when $B_1 = B_2$.* (ii) *The partition problem can be solved in pseudo-polynomial time: that is, there is an algorithm using 'dynamic programming' which solves the problems in about nB operations, where $B = \min(B_1, B_2)$. Therefore if B is small then the problem is easy.* (iii) *The 3-partition problem is* **NP***-complete, even when the lengths B_i are small ($\leq 2^{16}n^4$).*

Corollary 4.6 *Spot-scheduling is* **NP***-complete, even where the breaks are 'small' (polynomially bounded).*

Proof. Given a spot-scheduling problem Σ and a possible solution schedule S, it is easy (2.2) to check that S satisfies the constraints (4.2). Therefore the problem is in **NP**.

On the other hand, the 3-partition problem (4.3) is easily reduced (2.8) to a spot-scheduling problem: given a 3-partition problem, associate each of the numbers ℓ_j with a spot of length ℓ_j and colour c_j, where all the colours are distinct. Thus spot-scheduling is **NP**-complete. □

However, this is not the end of the story, since in practice breaks are at most 3 minutes long.

Lemma 4.7 *Let B be a fixed positive integer. Then the restricted form of multi-partition (4.3) where all break-lengths are bounded by B, is easy.*

Proof. For $1 \leq i \leq B$ there are finitely many partitions of i into a sum of positive integers. That is, there are finitely many lists of positive integers $x_1, \ldots, x_k$ whose 'weight' $\sum x_j$ is $\leq B$. Let

$$p_1, \ldots, p_L$$

be an enumeration of these partitions, arranged in non-increasing order of weight.

Suppose that a bounded problem instance ℓ_j, $1 \leq j \leq n$; B_i, $1 \leq i \leq m$, is given, where the spots and the breaks are also arranged in non-increasing order of length. Suppose that a solution $L_1, \ldots, L_m$ exists. Each sublist L_i defines a partition of B_i, and hence corresponds to one of the partitions p_r. Let the m partitions be placed in blocks with non-decreasing indices:

$$p_1, p_1, \ldots, p_1, p_2, \ldots p_2, p_3 \ldots p_L$$

[6] For example, n could be the number of bits used in a description of the input. We usually take some more natural measure: for example, in spot-scheduling, n could be the number of spots plus the number of breaks.

Some of the blocks may be empty. A valid schedule corresponds to one of these arrangements in which the i-th partition in the list (there are m) has weight $\leq B_i$, and if all the numbers in all the partitions are arranged in non-increasing order, we get the spot-lengths $\ell_1, \ldots, \ell_n$.

According to the well-known 'stars and bars' counting trick, each such arrangement can be represented by adding $L-1$ markers between m objects, where the markers separate adjacent blocks. Therefore there are

$$\binom{m+L-1}{L-1}$$

such arrangements: a polynomial in m of degree $L-1$, which depends on B, the bound on break-lengths, but B is fixed.

To solve the problem, all of these arrangements can be inspected in turn, until one is found, if it exists, which defines a valid schedule. This can be done in polynomial time. $\square$

It is not clear whether the above lemma extends to general spot scheduling with colour constraints (4.2). We will prove a related **NP**-completeness result. The result is stated below, followed by a definition of colour restriction.

Theorem 4.8 *The related problem of spot-scheduling with colour restrictions is* **NP***-complete, even with bounded break-lengths. (Proof in 5.19.)*

(4.9) Colour restrictions and spot fixing. By this we mean that certain breaks are closed off to certain colours. This can arise in two ways. First, there are restrictions on the advertising times of certain products, such as cigarettes and alcohol during children's programmes.

Second, advertisers can, at an extra charge, get their products advertised in certain desirable breaks, such as confectionery during children's programmes. We call such spots *fixed spots*. They have the effect of reducing the break length and excluding other spots of that colour.

Formally, a scheduling problem with colour restrictions is presented by specifying with each break a list of colours which may be scheduled in that break. With the ith break there is associated a list $L(i)$ of colours, and the constraints (4.2) are extended by requiring

- **Colour restriction:** if $S(j) = i$ then $c(j) \in L(i)$.

5 Proof of Theorem 4.8

(5.1) The proof of Theorem 4.8 is in two parts. First we prove the result with polynomially-bounded break lengths, and then extend the result to bounded break lengths. In order to prove Theorem 4.8 we introduce another **NP**-complete problem, a restricted form of 3SAT (3.4).

Definition 5.2 *A* balanced 3CNF Φ *is a 3CNF, that is a CNF in which every clause has 3 literals, in which every Boolean variable X occurs exactly as often* *as its complement $\overline{X}$.*

For example,

$$(A \vee B \vee C) \ \& \ (\overline{A} \vee B \vee C) \ \& \ (A \vee \overline{B} \vee \overline{C}) \ \& \ (\overline{A} \vee \overline{B} \vee \overline{C})$$

is a balanced 3CNF.

Lemma 5.3 *Balanced 3SAT, the satisfiability problem for balanced 3CNFs, is* **NP***-complete.*

Proof. Given an instance Φ of Balanced 3SAT and a truth-assignment θ, it is easy (2.2) to check that θ satisfies Φ, so Balanced 3SAT is in **NP**.

3SAT is known to be **NP**-complete (3.4), so by invoking Corollary 2.12 it is enough to show that

$$\text{3SAT} \leq \text{Balanced 3SAT}.$$

Let Ψ be any 3CNF. We need to construct a balanced 3CNF Φ which is satisfiable if and only if Ψ is. Φ is an extension of Ψ, where new Boolean variables and clauses are added if necessary.

Let L be a literal occurring in Ψ, $\overline{L}$ its complement.[7] Let

$$e = (\text{no. of occurrences of } L \text{ in } \Psi) - (\text{no. of occurrences of } \overline{L}).$$

Without loss of generality, $e \geq 0$. If $e > 0$ we add new Boolean variables and clauses, with e new occurrences of $\overline{L}$.

The easy case is where $e \geq 2$. Introduce new Boolean variables $V_1, \cdots, V_e$, and add the clauses

$$(\overline{L} \vee V_1 \vee \overline{V_2}) \ \& \ (\overline{L} \vee V_2 \vee \overline{V_3}) \ \& \ \cdots \ \& \ (\overline{L} \vee V_{e-1} \vee \overline{V_e}) \ \& \ (\overline{L} \vee V_e \vee \overline{V_1})$$

to Φ. The Boolean variables V_j occur nowhere else in Φ. A truth assignment θ which makes all these new variables V_j true satisfies all these new clauses, whether $\theta(L)$ is true (1) or false (0).

Otherwise $e = 1$. Introduce new Boolean variables V_1, V_2, and V_3, and new clauses

$$(\overline{L} \vee V_1 \vee \overline{V_2}) \ \& \ (L \vee V_2 \vee \overline{V_3}) \ \& \ (\overline{L} \vee V_3 \vee \overline{V_1})$$

to Φ. Again, a truth-assignment which makes all these new variables true satisfies the new clauses, whatever value it assigns to L.

This completes the construction of Φ. By construction, Φ is a balanced 3CNF. Since every clause in Ψ is also a clause in Φ, a truth-assignment satisfying Φ will also satisfy Ψ. Conversely, a truth-assignment which satisfies Ψ can be extended to one satisfying Φ, simply by making all new Boolean variables true. Therefore Φ is satisfiable iff Ψ is. Construction of Φ from Ψ is easy (2.2), so 3SAT $\leq$ Balanced 3SAT as asserted. □

(5.4) In order to prove Theorem 4.8, it is enough to show that

[7] If $L = \overline{X}$ then $\overline{L} = X$.

Balanced 3SAT $\leq$ Spot-scheduling with colour restrictions.

The construction is quite direct. Until the end of this section, Φ is a balanced 3CNF, and Σ will be a spot-scheduling problem (with colour restrictions) which we derive from Φ.

(5.5) The table below summarises the correspondences between Φ and Σ, and between a truth-assignment θ and a schedule S, with explanations following.

Φ, θ	Σ, S
Boolean variable X	Boolean colours c_X and $c_{\overline{X}}$
Boolean variable X	Selection colours d_X and $d_{\overline{X}}$
Clause $(L \vee M \vee N)$	Clause break allowing only colours c_L, c_M, c_N, length 5
Boolean variable X occurring k times in Φ (so does $\overline{X}$)	k surplus breaks allowing only colours c_X and $c_{\overline{X}}$, length 1
Boolean variable X occurring k times in Φ	$4k$ literal spots: k each of colours c_X or $c_{\overline{X}}$ and length 1 or 2.
Boolean variable X occurring k times in Φ	$2k$ selection spots of colours d_X and $d_{\overline{X}}$ and even length from $2k$ to $4k-2$
Boolean variable X occurring k times in Φ	$2k+1$ selection breaks of even length from $2k$ to $4k-2$
Boolean variable X occurring k times in Φ, $\theta(X)=1$	S maps k unit c_X spots and k double $c_{\overline{X}}$ spots to clause breaks, k double c_X spots to selection breaks, k unit $c_{\overline{X}}$ spots to surplus breaks
Boolean variable X occurring k times in Φ, $\theta(X)=0$	vice-versa

(5.6) Boolean colours and selection colours. For every Boolean variable X, the scheduling problem Σ will include spots of four 'colours,' namely

- 'Boolean' colours c_X and $c_{\overline{X}}$. Spots of these colours correspond to literals X and $\overline{X}$ and will be called *literal spots.*
- 'Selection' colours d_X and $d_{\overline{X}}$. Spots of these colours will be called *selection spots.*

  A truth-assignment selects the truth-value 1 or 0 for X; the selection colours allow a consistent selection of literal spots in the schedule.

Breaks are of three kinds

- 'Clause' breaks
- 'Selection' breaks
- 'Surplus' breaks

(5.7) The literal spots in Σ. Let X be a Boolean variable occurring k times in Φ. Since Φ is balanced, $\overline{X}$ also occurs k times in Φ.

The scheduling problem Σ contains $4k$ 'literal spots' with colours c_X and $c_{\overline{X}}$. The idea will become clear when the clause breaks are discussed. Σ contains

- k literal spots of length 1 and colour c_X (respectively, $c_{\bar{X}}$), called *unit spots.*
- k literal spots of length 2 and colour c_X (respectively, $c_{\bar{X}}$), called *double spots.*

(5.8) Clause breaks. For every clause $L \vee M \vee N$ in Φ, there is a corresponding 'clause break' b in Σ. Colour restrictions are used to ensure that only spots with the 'Boolean' colours c_L, c_M, and c_N, can be scheduled in b, which has length 5.

(5.9) Interim discussion. Suppose that Φ is satisfiable. Let θ be a truth-assignment satisfying Φ. Using θ, a schedule S satisfying Σ can be defined. S will map exactly half of the literal spots into clause breaks: namely, for any literal L, suppose (without loss of generality) that $\theta(L) = 1$. S assigns all the *unit* literal c_L-spots, and all the *double* literal $c_{\overline{L}}$ spots, to clause breaks corresponding to those clauses containing L and $\bar{L}$.

The remaining literal spots are mapped to the selection and surplus breaks, discussed below.

Since each clause contains a true literal, at least one of the literal spots (there are 3) mapped to the corresponding clause break is a unit spot. Therefore the total length of these spots is at most 5, so the size and colour constraints are satisfied in the clause breaks, at least.

(5.10) Surplus breaks. Again let L be a literal occurring k times in Φ. Σ contains k 'surplus breaks' corresponding to L. Each can only accept spots of colours c_L and $c_{\overline{L}}$, and has length 1.

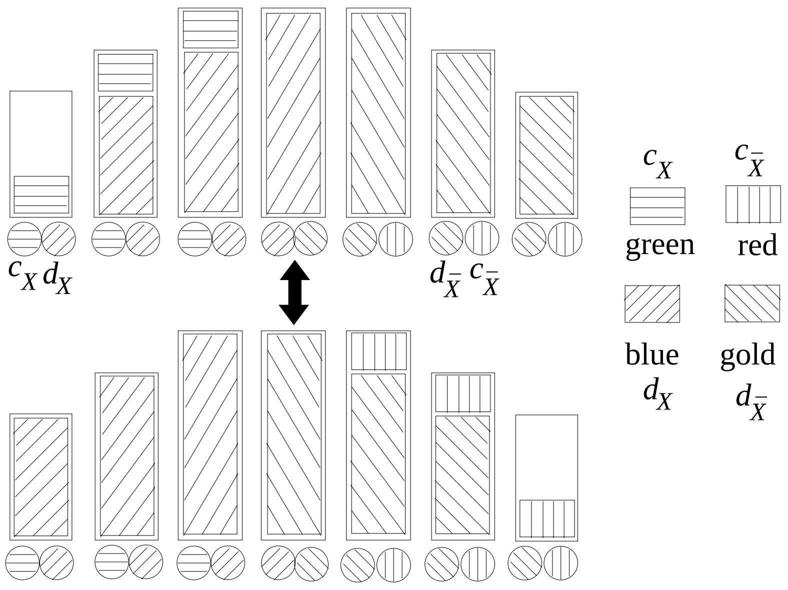

Fig. 1. selection subsystem where $k = 3$, c_X is green, d_X blue, $c_{\overline{X}}$ red, and $d_{\overline{X}}$ gold. Breaks and spots are rectangular. The coloured circles below a break show which colours are allowed in that break.

Suppose again that θ is a truth-assignment for Φ, $\theta(L) = 1$, and S is a corresponding schedule for Σ which has been partially described. It maps all unit literal c_L spots, and all double $c_{\overline{L}}$ spots, to clause breaks. It maps the unit $c_{\overline{L}}$ spots to these surplus breaks, and the double c_L spots to the selection breaks for L, discussed next.

(5.11) Selection spots and breaks. Let X be a Boolean variable occurring k times in Φ (as does $\overline{X}$). There is a subsystem of Σ involving k *selection spots* of colour d_X (respectively, $d_{\overline{X}}$), and even lengths in an arithmetic progression from $2k$ to $4k - 2$. This gives $2k$ selection spots for X.

Correspondingly, there is a subsystem of *selection breaks* for X. It involves k breaks of even length from $2k$ to $4k - 2$, each one allowing only the colours c_X and d_X (respectively, $c_{\overline{X}}$ and $d_{\overline{X}}$). This gives $2k$ breaks altogether. The selection subsystem has one more break of length $4k - 2$, allowing only the colours d_X and $d_{\overline{X}}$. See Figure 1.

There are k selection spots of colour d_X (respectively, $d_{\overline{X}}$), and $k + 1$ breaks (selection breaks) where they can be scheduled. Referring to Figure 1, consider the 'middle' break which can accept just blue and gold spots. If a blue spot is scheduled in that break, then no gold spot can be scheduled there (the shortest blue and gold spots have length $2k$ and the middle break has length $4k-2$). This means that only k breaks are available for scheduling gold spots and the gold spots fill those breaks completely. As a result, no red spots can be scheduled in those breaks, but it is possible to schedule k double green spots. More formally,

Lemma 5.12 *Let X be a Boolean variable occurring k times in Φ, as does*

$\overline{X}$.

(i) *It is possible to schedule k double c_X (respectively, $c_{\overline{X}}$) spots in the selection breaks for X, together with the selection spots for X.*

(ii) *If a valid schedule S maps* some *(literal) c_X spot (respectively, $c_{\overline{X}}$ spot) into these selection breaks, it cannot map* any *$c_{\overline{X}}$ (respectively, c_X) spot into these selection breaks.*

Proof. (i) Easy — see Figure 1.

(ii) Suppose, without loss of generality, that a c_X spot is scheduled in a selection break b. There are k such breaks where the spot can be scheduled: suppose b is the jth in ascending order of length. Its length is $2(k+j-1)$. There are $k-j+1$ selection spots of colour d_X and length $\geq 2(k+j-1)$, and since b contains a c_X spot of nonzero length, it cannot hold any of them. This means there are $k-j+1$ selection breaks sufficiently large to hold these spots. S must map one to each of them, so the 'middle break,' which allows d_X and $d_{\overline{X}}$ spots, contains a d_X spot. This break has length $4k-2$ and the shortest d_X spot has length $2k$, so there is insufficient room left to hold *any* $d_{\overline{X}}$ spot.

Therefore all the k $d_{\overline{X}}$ spots must be mapped to the remaining k selection breaks: they fill them completely, leaving no room for any $c_{\overline{X}}$ spots. □

Lemma 5.13 *If Φ is satisfiable then Σ has a valid schedule.*

Proof. Let θ be a truth-assignment satisfying Φ. A valid schedule S is defined as follows. Let X be a Boolean variable occurring k times in Φ (so does $\overline{X}$). There are $6k$ spots associated with X: $4k$ literal spots and $2k$ selection spots. S schedules these as follows.

- If $\theta(X) = 1$ then S maps the unit c_X spots and the double $c_{\overline{X}}$ spots into the $2k$ clause breaks available for these colours.

  It maps the unit $c_{\overline{X}}$ spots to the surplus breaks for X, and the double c_X spots plus the selection spots to the selection breaks for X (Lemma 5.12 (i)).
- If $\theta(X) = 0$ then X and $\overline{X}$ interchange rôles and the mapping is defined symmetrically.

In view of Lemma 5.12 and paragraph 5.9, S is a valid schedule for Σ. □

The following definition is only to clarify the proof of Corollary 5.16 below. It reflects the idea that the existence of single (respectively, double) c_X-spots in clause breaks mean X is true (respectively, false) (5.9, 5.13).

Definition 5.14 *Let X be a Boolean variable in Φ. A valid schedule S for Σ is X-*consistent *if S assigns only single c_X-spots and double $c_{\overline{X}}$-spots to clause breaks, or vice-versa.*

Lemma 5.15 *If Σ has a valid schedule S then it has a valid schedule which is X-consistent for every X.*

Proof. Let X be a Boolean variable such that S is not X-consistent. Without loss of generality, it maps both single and double c_X-spots to clause breaks.

Suppose there are k occurrences each of X and of $\overline{X}$ in the clauses in Φ. There are k double c_X-spots, and not all are mapped to clause breaks. The surplus breaks are too short to contain them, so they must be mapped to selection breaks. From Lemma 5.12 (ii), no $c_{\overline{X}}$-spot is mapped to any selection break, nor can any double $c_{\overline{X}}$-spot be mapped to a surplus break, so all double $c_{\overline{X}}$-spots are mapped to clause breaks and all single $c_{\overline{X}}$-spots are mapped to surplus breaks.

Therefore all c_X-spots are mapped to clause breaks and selection breaks, since all surplus breaks for X are contain $c_{\overline{X}}$-spots.

The double c_X-spots in clause breaks can be swapped with the single c_X-spots in selection breaks while preserving the length- and colour-constraints.

Do this for all Boolean variables X for which S is not X-consistent; this yields a valid schedule S' which is X-consistent for every X. □

Corollary 5.16 *If Σ has a valid schedule then Φ is satisfiable.*

Proof. Assuming Σ has a valid schedule S, we can assume that S is X-consistent for every X. Define a truth-assignment θ for Φ as follows: for every Boolean variable X,

- if S assigns single c_X-spots and double $c_{\overline{X}}$-spots to clause breaks, then $\theta(X) = 1$.
- If S assigns single $c_{\overline{X}}$-spots and double c_X-spots to clause breaks, then $\theta(X) = 0$.

Since S is X-consistent these cases are mutually exclusive.

Claim that one or other case must apply. Let k be the number of occurrences of X, and also of $\overline{X}$, in the clauses in Φ. There are $4k$ c_X- and $c_{\overline{X}}$-spots overall, with $5k$ available breaks to place them: $2k$ clause breaks, k surplus breaks, and $2k$ selection breaks, but the selection breaks cannot take both c_X and $c_{\overline{X}}$-spots (Lemma 5.12 (ii)), so they have room for only k of the spots. Therefore all the $2k$ available clause breaks are filled with c_X- and $c_{\overline{X}}$-spots. This proves the claim.

It also shows that for every clause C in Φ, $C = L \vee M \vee N$, say, the corresponding clause break contains single or double spots of colours c_L, c_M, and c_N under S. At least one of them, the c_L-spot, say, has length 1. Then $\theta(L) = 1$, so θ satisfies C. Since C is arbitrary, θ satisfies Φ. □

(5.17) Proof of weak form of Theorem 4.8 (that spot-scheduling with colour restrictions is **NP**-complete, with polynomial-size breaks).

Given a problem instance Σ of a spot-scheduling with colour restrictions, it is obviously easy to verify that a given schedule S of Σ is valid. Therefore the problem is in **NP**.

Given a balanced 3CNF Φ, we have described a spot-scheduling problem (with colour restrictions) Σ which has a solution if (Lemma 5.13) and only if (Corollary 5.16) Φ is satisfiable. Σ is obviously easy to construct from Φ. Therefore

$$\text{Balanced 3SAT} \leq \text{Spot-scheduling with colour restrictions.}$$

The former is **NP**-complete (5.3) and the latter is in **NP**, so it is **NP**-complete (2.12). □

To complete our proof, it is enough to show the following

Lemma 5.18 *A restricted form of BALANCED 3SAT, in which each literal occurs at most 5 times, is* **NP***-complete.*

Proof. The problem is in **NP** by a routine argument which we omit.

Let Φ be a balanced 3CNF in which a Boolean variable A occurs $k > 5$ times (as must its complement $\overline{A}$). Create a new balanced 3CNF Φ' in which corresponding to A and its complement there are k variables $A_1, \ldots, A_k$ (and their complements), each one occurring in a location corresponding to A (or $\overline{A}$). Now Φ' has only one occurrence of each A_i and $\overline{A_i}$.

It is enough to extend Φ' by clauses ensuring the logical equivalence of the variables $A_1, \ldots A_k$. For $1 \leq i < k$, add the clauses

$$(A_i \vee \overline{A_{i+1}} \vee X_i) \ \&(A_i \vee \overline{A_{i+1}} \vee \overline{X_i}) \ \&(\overline{A_i} \vee A_{i+1} \vee X_i) \ \&(\overline{A_i} \vee A_{i+1} \vee \overline{X_i}) \quad (1)$$

where the X_i are $k-1$ new variables.

For each i, the expression (1) is balanced, so Φ' is balanced. Any truth-assignment to (1) makes the expression true iff and only if it makes A_i and A_{i+1} both true or both false. Thus the only truth-assignments satisfying Φ' make the A_i all true or all false. It follows easily that Φ is satisfiable iff Φ' is.

Each X_i and its complement occur twice Φ', and A_i and $\overline{A_i}$ occur 3 times if $i = 1$ or $i = k$ and 5 times if $2 \leq i \leq k-1$.

Repeat this procedure for every variable occurring more than 5 times in Φ, obtaining a balanced 3CNF Ψ which is satisfiable iff Φ is.

Construction of Ψ from Φ is easy, and Ψ has the desired form (balanced 3CNF in which no literal occurs more than 5 times). Therefore the satisfiability problem for 3CNFs of the desired form is **NP**-complete. □

(5.19) Proof of Theorem 4.8. The proof of the weak form (5.17) argued that satisfiability of a balanced 3CNF Φ is easily reduced (2.8) to a spot-scheduling problem Σ.

By Lemma 5.18, we can restrict our attention to those balanced 3CNFs Φ in which every literal occurs at most 5 times.

With this assumption, referring to the table 5.5, the parameter k (number of occurrences of a literal) is at most 5. The selection breaks have length at most $4k-2$, i.e., at most 18. The other breaks have lengths 1 and 5. Therefore the longest break is 18 units long. □

6 Scheduling unit-length spots is easy

We consider spot-scheduling problems where all spots have the same length, without loss of generality, 1. First we consider the on-line or dynamic problem, where the breaks are fixed but the spots are booked individually. The schedule should accommodate all spots currently booked, and signal when no schedule is possible. *Spot dispersal* (see [ii]) means re-scheduling existing spots to make room for an incoming spot.

Spot dispersal works with or without colour restrictions and/or spot fixing (4.9).

Definition 6.1 *A break is* full *(or* filled*) under the schedule S if its length equals the total length of spots scheduled in the break.*

(6.2) Dispersal chains (See [ii]). Let S be a schedule solving a certain unit-length scheduling problem. A *dispersal chain* is a mixed sequence of distinct breaks and distinct colours, such as

$$b_0 \leftarrow \kappa_1 \leftarrow b_1 \leftarrow \kappa_2 \leftarrow \cdots \kappa_r \leftarrow b_r,$$

indicating (if $r > 0$) that S can be altered by moving a spot of colour κ_1 from b_1 to b_0, and so on, until finally a spot of colour κ_r is moved from b_r to b_{r-1}.

The break b_0 must not be full, but all the other breaks are full. The $(j+1)$st break must of course contain a non-fixed spot of colour κ_{j+1}, and b_j must not contain a spot of colour κ_{j+1} and must not exclude spots of that colour through colour restrictions.

Under these assumptions, the altered schedule satisfies the constraints (4.2, 4.9).

Lemma 6.3 *Suppose that S is a schedule for some spots, and a new non-fixed spot, coloured red, say, is to be scheduled. Let R be the set of breaks not assigned red spots by S and not excluding red spots under colour restrictions, and let N be the set of breaks not filled by S.*

Then the incoming red spot can be scheduled together with the the previous spots, if and only if a dispersal chain exists connecting a break in R to a break in N.

Proof. If: easy.

Only if: If $R \cap N$ contains a break b_0 then we can extend S by assigning the last red spot to that break (and $\{b_0\}$ is a dispersal chain).

So we assume $R \cap N = \emptyset$ and a schedule S' exists for all the spots, including the last red one. S' is not necessarily an extension of S.

We construct a labelled directed graph G reflecting the difference between S and S'. The nodes of G are, or correspond to, the breaks. Given distinct breaks b and c, if there exists a spot colour κ which S assigns to b but not c and S' assigns to c and not b, then G contains a directed edge from b to c, labelled with all such colours κ.

This gives a fragment of a possible dispersal chain: $\ldots c, \kappa, b \ldots$ where κ is the colour of one of the spots labelling the edge (c comes before b in the dispersal chain as the definitions require). In general, a simple path from R to N in G defines (in reverse order) a dispersal chain whose existence we are trying to prove. We shall attempt to construct such a simple path, from R to N in G.

If instead we discover a cycle in the graph, we use it to alter S', making it agree more closely with S. We then repeat the attempt.

S' schedules one more red spot than does S, so it must schedule a red spot in a break where S does not. This break belongs to R: $R \neq \emptyset$.

Beginning with a break in R, randomly generate a path (sequence of edges labelled with spots) in the graph until either (a) a node in N is reached or (b) some node is visited twice. The path can always be extended until either (a) or (b) becomes true, for the following reason.

Claim: if c is a break outside N (i.e., full under S) and either $c \in R$ or c has an incoming edge, then c has an outgoing edge.

To prove this, let ℓ be the length of c, so S assigns a set C of ℓ different colours to c. S' assigns to c some colour κ not in C: if $c \in R$, then take κ to be red; otherwise, κ is some colour labelling an incoming edge.

Therefore there is some colour $\lambda \in C$ which S' does not assign to c. But S' schedules every spot which S does, so there exists a break d to which S' but not S assigns the colour λ, and d has an outgoing edge labelled λ, proving the claim.

Therefore the path can always be extended until (a) or (b) holds. If case (a) is encountered before case (b), the path defines the desired dispersal chain and we are finished. Otherwise the graph G contains a simple directed cycle:

$$a \rightarrow^{s} b \rightarrow^{t} \cdots g \rightarrow^{v} a$$

This time, rather than altering S, we use this cycle to alter S'. By changing $S'(s)$ to a, $S'(t)$ to b, and so on until $S'(v) = g$, we get another valid schedule for all spots (including the last red one). The alterations preserve the colour constraints, since a did not already contain under S' a spot clashing with s, and so on.

The altered version agrees more closely with S, so if we repeat the construction of G we get a labelled directed graph with the same properties as before, but with fewer edges and/or edge-labels. Hence after a finite number of steps we obtain the desired kind of dispersal chain. □

(6.4) Finding a dispersal chain. Given a schedule S and a new spot of colour c, call it red, let B_0 consist of all breaks not filled by S. Let R consist of all breaks not assigned red spots by S and not excluding red spots.

Any break in $R \cap B_0$ is a trivial dispersal chain, so if the intersection is nonempty, stop. Otherwise, let K_1 consist of all colours κ for which B_0 contains a break not containing spots of colour κ and not excluding κ. Let B_1

consist of all breaks, not in B_0, containing non-fixed spots whose colours are in K_1.

In general, K_{i+1} consists of all colours κ, not in $\bigcup_0^i K_j$, for which B_i contains a break not containing spots of that colour and not excluding that colour. Let B_{i+1} consist of all breaks, not in $\bigcup_0^i B_j$, which contain non-fixed spots whose colours are in K_{i+1}.

Stop when a set B_i is generated which either intersects R or it (and all subsequent B_j and K_j) is empty.

Clearly this procedure is easy (polynomial time).

Lemma 6.5 *A break b is connected to a break in B_0 by a dispersal chain of length j if and only if $b \in B_i$ for some $i \leq j$. Also it is easy (2.2) to construct such a chain. (Proof by a straightforward induction on j.)* □

Theorem 6.6 *The problem of scheduling unit-length spots can be solved easily (i.e., in polynomial time) by spot dispersal, whether or not the problem has colour restrictions or fixed spots.* □

(6.7) The off-line unit-length scheduling problem without colour restrictions. If we defer the scheduling problem until all the spots have been received, and there are no fixed spots or colour restrictions, there is a more efficient method:

- Arrange the spots into groups according to colour: each group contains all spots of a given colour.
- Arrange the breaks in non-increasing order of length.
- Repeat the following for each group of spots:

  Schedule the spots in the group in a 'greedy' fashion that is, the first spot goes into the first break, the second into the second break, and so on, until all spots in the group are scheduled or there are no breaks left with free space available. In that case, stop and report 'problem unsolvable.'

  Otherwise, re-order the breaks according to non-increasing *residual* length (that is, break length minus length of spots scheduled in break) and continue with the next colour group.

(6.8) The above method can be implemented very efficiently (linear time, meaning the run-time is proportional to the input size) using linked lists. The *correctness* of the method is not obvious.

Lemma 6.9 *If a valid schedule exists then the method (6.7) will find one.*

Proof. Otherwise, consider an example where method schedules spots as far as a red spot, say, at which point there is no available break, but an alternative schedule exists, therefore a dispersal chain exists:

$$b_0 \leftarrow \kappa_1 \leftarrow b_1 \leftarrow \kappa_2 \leftarrow \cdots \kappa_r \leftarrow b_r, \tag{2}$$

and the following may be assumed.

(i) All breaks in the list except b_r contain a red spot.

(ii) All breaks in the list except b_0 are full.

(iii) For any $i \geq 1$, κ_i is scheduled in b_i but not in b_{i-1}. In particular, $\kappa_i \neq$ red.

(iv) For any $i \geq 1$, κ_i is the most recently scheduled colour in b_i which is not also in b_{i-1}: all more recently scheduled colours are in b_{i-1}.

(v) For any $i \geq 0$ and $j \geq i+2$, all colours scheduled in b_j are also in b_i. Otherwise, one could make a shorter dispersal chain which moves a spot from b_j to b_i. For the same reason one can assume that all colours κ_i are distinct.

In what follows, we shall be considering adjacent pairs of breaks in the list (2):

$$\ldots b_{i-1} \leftarrow \kappa_i \leftarrow b_i \ldots \tag{3}$$

Let P be the set of spot colours scheduled after κ_i in b_i. Since b_i is full, before κ_i was scheduled, b_i contained $1 + |P|$ unfilled units. Since b_{i-1} contains spots of all colours in P, (iv), before κ_i was scheduled it contained at least $|P|$ units of free space.

If we can show that b_{i-1} contained two extra free units besides these $|P|$, then we have a contradiction, because the algorithm 6.7 should have placed a spot of colour κ_i in b_{i-1} before b_i.

Claim 1: There are at least 3 breaks in the list (2), i.e., $r > 1$. From (i) and (ii), $r > 0$, so if the claim is false then $r = 1$.

Assume $r = 1$ and consider (3) with $i = r = 1$, P the set of colours scheduled in b_1 after κ_1. When κ_1 was scheduled, b_0 contained at least two extra free units: one for the red spot, which is not in b_1, (i), therefore not in P, and one that remains unfilled, (ii). Hence a contradiction, proving the claim.

Claim 2: κ_1 was scheduled *after* κ_2. Otherwise, it was scheduled before κ_2, since $\kappa_1 \neq \kappa_2$, (v). Consider (3) with $i = 1$, P as previously. Before κ_1 was scheduled, b_0 contained two extra free units: one for κ_2 (after κ_1, in b_0, (v), not in b_1, (iii), therefore not in P), and one that remains unfilled, (ii). Hence a contradiction, proving the claim.

Claim 3: For $1 \leq i < r$, κ_i was scheduled *after* κ_{i+1}. This is proved by induction, with Claim 2 as the base case. The inductive hypothesis is that κ_{i-1} was scheduled after κ_i. P is the set of colours scheduled in b_i after κ_i.

Assume that κ_{i+1} was also scheduled after κ_i. Again considering 3, before κ_i was scheduled there are two extra free units in b_{i-1}. One for κ_{i-1}: after κ_i, (induction), in b_{i-1} and not in b_{i-2}, (iii), hence not in b_i, (v), hence not in P. One for κ_{i+1}: after κ_i, (assumption), in b_{i-1}, (v), but not in b_i, (iii). Hence, a contradiction, proving the inductive step and Claim 3.

From Claim 3, κ_{r-1} was scheduled after κ_r, and $r \geq 2$ (Claim 1). Consider (3) with $i = r$, P the set of colours scheduled after κ_r in b_r. When κ_r was

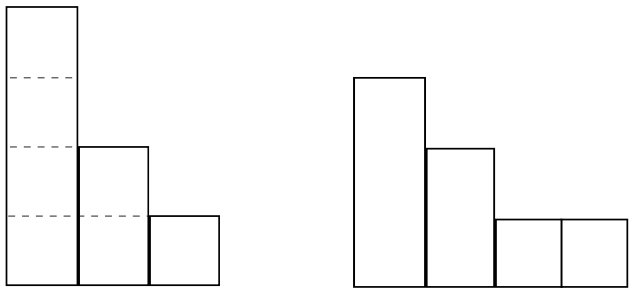

Fig. 2. A non-increasing sequence and its transpose.

scheduled, b_{r-1} contained at least two extra free units: one for a red spot, not in P, (i,ii), and one for κ_{r-1} (after κ_r, in b_{r-1} and not in b_{r-2}, (iii), hence not in b_r, (v), so not in P). Hence, a contradiction.

This contradiction shows that the dispersal chain (2) cannot exist: the method is correct. □

7 Another criterion for unit-length spots

In this section we establish an alternative test for solvability of the scheduling problem with unit-length spots. Unlike the previous section, the criterion (Theorem 7.7) is a formula that doesn't explicitly construct a schedule.

It requires the idea of 'domination' between non-increasing sequences.

Definition 7.1 (i) *A* non-increasing sequence *is a countable sequence* $\boldsymbol{x} = \{x_1, \ldots\}$ *of natural numbers, where* $x_j \geq x_{j+1}$ *for all* j*, and for some* j*,* $x_j = 0$*, so only finitely many components are nonzero.*

(ii) *Write* $\sum \boldsymbol{x}$ *for the sum* $\sum_{j\geq 1} x_j$*, necessarily finite.*

(iii) *One such sequence* $\boldsymbol{d} = \{d_i\}$ dominates *another* $\boldsymbol{c} = \{c_i\}$ *if for all* k*,*

$$\sum_{i=1}^k c_i \leq \sum_{i=1}^k d_i$$

(In particular, $\sum \boldsymbol{c} \leq \sum \boldsymbol{d}$*.)*

Lemma 7.2 *This relation is reflexive, transitive and anti-symmetric.*

Proof. In other words, (i) $\boldsymbol{x}$ always dominates itself, (ii) if $\boldsymbol{x}$ dominates $\boldsymbol{y}$ and $\boldsymbol{y}$ dominates $\boldsymbol{z}$, then $\boldsymbol{x}$ dominates $\boldsymbol{z}$, and (iii) if $\boldsymbol{x}$ dominates $\boldsymbol{y}$ and $\boldsymbol{y}$ dominates $\boldsymbol{x}$, then $\boldsymbol{x} = \boldsymbol{y}$.

These are straightforward. For part (iii), prove by induction that if, for all k, $\sum_{i=1}^k x_i = \sum_{i=1}^k y_i$, then for all k, $x_k = y_k$. □

Definition 7.3 *The* transpose $\boldsymbol{x}^T$ *of a non-increasing sequence* $\boldsymbol{x}$ *is the sequence* $\{t_1, \ldots\}$*, where for each* i*,* t_i *is the number of indices* j *such that* $x_j \geq i$*.*

(7.4) Figure 2 illustrates a non-increasing sequence and its transpose represented by bar-charts drawn on a square grid.

It is more illuminating to observe that t_1 is the width of the bottom row of (the bar chart representing) $\boldsymbol{x}$, t_2 the width of its second row, and so on.

Notice that the transpose of $\boldsymbol{x}^T$ is $\boldsymbol{x}$, and $\sum \boldsymbol{x}^T = \sum \boldsymbol{x}$ (7.1).

(7.5) Non-increasing sequences presenting a unit-length scheduling problem. Let us consider a unit-length spot scheduling problem. Suppose that the break-lengths are represented by a non-increasing sequence $\boldsymbol{b}$ and the colour frequency distribution among spots by a non-increasing sequence $\boldsymbol{c}$. (Thus c_i is the number of spots with the i-th colour, where the colours are arranged to make $c_i \geq c_{i+1}$.)

The following lemma is a simple case of the criterion of Theorem 7.7 below.

Lemma 7.6 *Given a unit-length scheduling problem $\boldsymbol{b}, \boldsymbol{c}$, if $\boldsymbol{c} = \boldsymbol{b}^T$ then the problem is solvable.*

Proof. Visualise the breaks $\boldsymbol{b}$ as a bar-chart on a square grid as in Figure 2. Put all c_1 spots of the first colour in the bottom row, all c_2 spots of the second colour in the second row, and so on. This is a valid schedule. □

Theorem 7.7 (domination criterion). *The unit-length scheduling problem with break-length distribution $\boldsymbol{b}$ and colour distribution $\boldsymbol{c}$ has a solution if and only if $\boldsymbol{b}^T$ dominates $\boldsymbol{c}$. (Proof occupies the rest of this section.)*

(7.8) Criterion is easy to apply. This criterion is easily checked. Indeed, the vectors $\boldsymbol{b}^T$ and $\boldsymbol{c}$ can be constructed in linear time (6.8).

The next lemma covers the 'only if' part.

Lemma 7.9 *The criterion in Theorem 7.7 is necessary for the given scheduling problem to be solvable.*

Proof. Suppose that a schedule exists. Write $\boldsymbol{w} = \{w_1, \ldots\}$ for $\boldsymbol{b}^T$.

For any k such that $c_k \neq 0$ consider the scheduled spots of the first k colours. Each break contains at most k of these spots. More precisely, there are at most kw_k of these spots in breaks of length $\geq k$ (w_k is the number of breaks of length $\geq k$: see Definition 7.3). There are at most $(k-1)(w_{k-1}-w_k)$ of them in breaks of length $k-1$, since $w_{k-1} - w_k$ is the number of breaks of length $k-1$; there are at most $(k-2)(w_{k-2} - w_{k-1})$ of them in breaks of length $k-2$, and so on.

Therefore

$$c_1 + c_2 + \cdots c_k \quad \leq \quad kw_k + \sum_{i=1}^{k-1} i(w_i - w_{i+1}).$$

This sum can be written as

$$(k-(k-1))w_k + ((k-1)-(k-2))w_{k-1} + \cdots (2-1)w_2 + w_1,$$

so

$$\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} w_i.$$

This holds for all k such that $c_k > 0$, so clearly it holds for all k: $\boldsymbol{w}$ dominates $\boldsymbol{c}$. In other words, the criterion of Theorem 7.7 is satisfied. $\square$

In order to prove the 'if' part of Theorem 7.7, we start with a related solvable problem, given by Lemma 7.6, and alter the problem with a series of 'levelling moves' (described later) until we arrive at the original problem.

Lemma 7.10 *Suppose a non-increasing sequence $\boldsymbol{w} = \{w_i\}$ dominates another one $\boldsymbol{c} = \{c_i\}$.* (i) *Then there exists another sequence $\boldsymbol{w'} = \{w'_i\}$ which $\boldsymbol{d}$ dominates and which dominates $\boldsymbol{c}$, such that $\sum \boldsymbol{w'} = \sum \boldsymbol{c}$.*

Also, (ii) *for each i, $w'_i \leq w_i$, and* (iii) *for each i, $b'_i \leq b_i$ where $\{b_i\} = \{w_i\}^T$ and $\{b'_i\} = \{w'_i\}^T$.*

Proof. (i) It follows from the definition of 'dominates' that $\sum_{i\geq 1} c_i \leq \sum_{i\geq 1} w_i$. Suppose the inequality is strict.

Write C for $\sum \boldsymbol{c}$, and let $m \geq 0$ be maximal subject to the condition

$$\sum_{i=1}^{m} w_i \leq C.$$

Define

$$w'_k \quad = \begin{cases} w_k, & k \leq m \\ C - \sum_{i=0}^{m} w_i, & k = m+1 \\ 0, & k > m+1 \end{cases}$$

Obviously (ii) holds, and therefore $\boldsymbol{w}$ dominates $\boldsymbol{w'}$.

For any $k \leq m$, $\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} w'_i$.

If $k \geq m+1$ then $\sum_{i=1}^{k} w'_i = C \geq \sum_{i=1}^{k} c_i$, so $\boldsymbol{w'}$ dominates $\boldsymbol{c}$. This proves (i).

It is easy to show that (ii) implies (iii) (Definition 7.3). $\square$

Definition 7.11 *A* levelling move *is a procedure applied to a non-increasing sequence (here representing a colour distribution), making the sequence more even as follows. The old and new sequences are identical except for two colours appearing r times and b times (with $r \geq b+2$) in the old sequence, and $r-1$ and $b+1$ times, respectively, in the new sequence.*

It is necessary that $r \geq b+2$ to ensure that the new sequence is also non-increasing.

Lemma 7.12 *Given a unit-length scheduling problem with breaks $\boldsymbol{b}$ and colours $\boldsymbol{c}$, if the problem is solvable, and $\boldsymbol{c}$ is altered by a series of levelling moves, then the new problem is also solvable.*

Proof. Consider the case of a single levelling move. Suppose in the problem one colour (let us say red) occurs r times, and another colour (let us say blue) occurs b times, and $r \geq b+2$. Then the related problem in which red only appears $r-1$ times while blue appears $b+1$ times is also solvable: given a valid schedule for the first problem, there must be at least one break where red appears without blue appearing. Change the colour of the red spot to blue, producing a valid schedule for the altered problem.

The general result follows by induction. □

Lemma 7.13 *Suppose a non-increasing sequence $\boldsymbol{d'}$ is obtained from another, $\boldsymbol{d}$, by a series of levelling moves. Then $\sum \boldsymbol{d} = \sum \boldsymbol{d'}$, and $\boldsymbol{d}$ dominates $\boldsymbol{d'}$.*

Proof. Easy to show for a single move, and follows trivially by induction, using Lemma 7.2. □

Lemma 7.14 *Let $\boldsymbol{c}$ and $\boldsymbol{d}$ be two non-increasing sequences, where $\sum \boldsymbol{c} = \sum \boldsymbol{d}$ and $\boldsymbol{d}$ dominates $\boldsymbol{c}$. Then one can obtain $\boldsymbol{c}$ from $\boldsymbol{d}$ by a series of levelling moves.*

Proof. Without loss of generality $\boldsymbol{c} \neq \boldsymbol{d}$. Let k be the smallest positive integer such that $c_k \neq d_k$. Then $\sum_{i=1}^{k} d_i \neq \sum_{i=1}^{k} c_i$, so $d_k > c_k$. Since $\sum \boldsymbol{c} = \sum \boldsymbol{d}$, $\sum_{k+1}^{\infty} c_j > \sum_{k+1}^{\infty} d_j \geq 0$, so $d_k > c_k \geq c_{k+1} > 0$, i.e., $d_k \geq 2$.

The sequence $\boldsymbol{d}$ may contain several entries equal to d_k: let ℓ be the largest index such that $d_\ell = d_k$. Then $d_{\ell+1} < d_\ell$. Let m be the smallest index $> \ell$ such that $d_m \leq d_\ell - 2$. It exists because $d_\ell \geq 2$.

Let $\boldsymbol{d'}$ be like $\boldsymbol{d}$ except $d'_\ell = d_\ell - 1$ and $d'_m = d_m + 1$. Then $\boldsymbol{d}$ dominates $\boldsymbol{d'}$ and $\sum \boldsymbol{d} = \sum \boldsymbol{d'}$ (Lemma 7.13).

To show that $\boldsymbol{d'}$ dominates $\boldsymbol{c}$, observe that for $1 \leq r < k$, $c_r = d'_r$, and for $k \leq r < m$, $c_r \leq c_k \leq d_k - 1 \leq d'_r$, i.e., for $1 \leq r < m$, $c_r \leq d'_r$, so $\sum_{i=1}^{r} c_i \leq \sum_{i=1}^{r} d'_i$. For $r \geq m$, $\sum_{i=1}^{r} d_i = \sum_{i=1}^{r} d'_i$, so $\sum_{i=1}^{r} c_i \leq \sum_{i=1}^{r} d'_i$.

Thus $\boldsymbol{d}$ dominates $\boldsymbol{d'}$ which dominates $\boldsymbol{c}$. If $\boldsymbol{d'} \neq \boldsymbol{c}$ perform another levelling move.

This series of levelling moves will produce a series of distinct (Lemma 7.2 (iii)) sequences $\boldsymbol{x}$. The series is finite since $\sum \boldsymbol{x} = \sum \boldsymbol{d}$ for all these sequences $\boldsymbol{x}$. Eventually $\boldsymbol{c}$ must be produced. □

Corollary 7.15 *The criterion of Theorem 7.7 is sufficient.*

Proof. Given a problem instance $\boldsymbol{b}, \boldsymbol{c}$, let $\boldsymbol{w} = \boldsymbol{b}^T$. Suppose the criterion holds, i.e., $\boldsymbol{w}$ dominates $\boldsymbol{c}$.

Applying Lemma 7.10 we obtain another sequence $\boldsymbol{w'}$ which is dominated by $\boldsymbol{w}$ and which dominates $\boldsymbol{c}$, with $\sum \boldsymbol{w'} = \sum \boldsymbol{c}$.

Let $\boldsymbol{b'} = \boldsymbol{w'}^T$. By Lemma 7.6 the problem $\boldsymbol{b'}, \boldsymbol{w'}$ is solvable.

Since $\sum \boldsymbol{w'} = \sum \boldsymbol{c}$, by Lemma 7.14 one can obtain $\boldsymbol{c}$ from $\boldsymbol{w'}$ by a series of levelling moves. It follows from Lemma 7.12 that the problem $\boldsymbol{b'}, \boldsymbol{c}$ is solvable.

But $b'_i \leq b_i$ for all i (Lemma 7.10 (iii)), so any solution to $\boldsymbol{b'}, \boldsymbol{c}$ also solves $\boldsymbol{b}, \boldsymbol{c}$, so the latter is solvable, as required. □

Lemma 7.9 and Corollary 7.15 complete the proof of Theorem 7.7.

8 Concluding remarks and open problems

Our results about spot-scheduling may be summarised as follows.

(i) Spot-scheduling is **NP**-complete, even when the break-lengths are polynomially bounded (reduction to multi-partition problem). This is true whether or not colour restrictions are allowed.

(ii) Spot-scheduling is **NP**-complete in the presence of colour restrictions, even when the break-lengths are bounded by 18 (4.8). The corresponding multi-partition problem is 'easy' (Lemma 4.7).

(iii) Spot-scheduling with unit-length spots is easy, with or without colour restrictions (6.6).

(iv) The off-line scheduling problem for unit-length spots, without colour restrictions, is solvable in linear time (6.8).

(v) There is a simple criterion (7.7) for unit-length spot-scheduling problems, which can be checked in linear time.

(8.1) One open problem is to design more efficient off-line methods which can handle colour restrictions, but the following is probably more difficult:

Problem 8.2 *What is the complexity of scheduling spots in bounded-length breaks without colour restrictions?*

Whether this is easy, like the bounded multi-partition problem, or **NP**-complete, as holds with colour restrictions, or somewhere between the two, remains to be seen.

9 Acknowledgement

The authors are indebted to an anonymous referee whose careful and perceptive review revealed several errors in an earlier draft.

10 References

(i) A.R. Brown (1969). Selling television time: an optimisation problem. *Computer Journal* **12:3** (August 1969).

(ii) A. R. Brown (1971). *Optimal packing and depletion.* Macdonald / American Elsevier Computer Monographs.

(iii) V. Chvátal (1980). *Linear Programming.* W.H. Freeman.

(iv) S. Cook (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd ACM symposium on theory of computing*, 151–158.

(v) Michael Garey and David S. Johnson (1979). *Computers and intractability: a guide to the theory of* **NP***-completeness.* W.H. Freeman.

(vi) Narendra Karmakar (1984). A new polynomial-time algorithm for linear programming. *Combinatorica* **4:4** 373–395.

(vii) Richard Karp (1972). Reducibility among combinatorial problems. *Complexity of computer computations*, ed. R. E. Miller and J. W. Thatcher, 85–103.