The complexity of scheduling TV commercials

Klemens Hägele *
Formal Methods Group, Computer Science, Trinity College, Dublin 2, Ireland
Colm Ó Dúnlaing
Mathematics, Trinity College, Dublin 2, Ireland
Søren Riis
Queen Mary, University of London

October 13, 2000

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 **NP**-complete, even when the breaks are bounded (at most 10 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 groups,' and products within the same clash group should not be advertised in the same break.[1]

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 [1,2] investigated this problem, but we know of no other research into the problem. It is certainly of interest in the context of **NP**-completeness.

*e-mail: khaegele@gmx.net, odunlain@maths.tcd.ie, smriis@dcs.qmw.ac.uk

[1]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.

This paper presents our discoveries 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 when the input has size n (suitably measured), the time to compute the result is bounded by some polynomial in n.

(2.2) 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.

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

$$AX \leq B$$

where A is a matrix an B a column-vector. Dantzig's Simplex Method of 1947 [3] 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 [6] is another polynomial-time 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 (2.10): strong evidence that no efficient method exists.

(2.4) Definition of NP. The class **NP** is, strictly speaking, the class of languages which can be accepted in nondeterministic polynomial time on a nondeterministic Turing machine. See [5] for a detailed survey of the subject.

(2.5) Definition *A computational task is 'easy' if it can be done in polynomial time. See 2.1.*

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

(2.6) Definition NP *is the class of 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 make the expression 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 (2.10,3.3). The satisfiability problem is important and will be discussed further in the next section.

(2.8) Definition *Given two computational problems X and Y, write $X \leq Y$ if X can be easily reduced to Y.*

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

(2.9) Corollary *If X is difficult and $X \leq Y$ then Y is difficult. (Proof easy.)* ∎

(2.10) Definition *A problem Y is* **NP**-complete *if* (i) $Y \in \mathbf{NP}$ *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.

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

(2.12) Definition P *is the class of 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

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

- 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, 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

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

then

$$\theta(\Phi) \quad = \quad (0 \vee 0 \vee 1) \ \& \ (0 \vee 0 \vee \overline{1}) \ \& \ (\overline{0}) \ \& \ (0 \vee \overline{0}) \quad =$$
$$(0 \vee 0 \vee 1) \ \& \ (0 \vee 0 \vee 0) \ \& \ (1) \ \& \ (0 \vee 1) \quad = 1 \ \& \ 0 \ \& \ 1 \ \& \ 1 \quad = \quad 0$$

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.

(3.3) Proposition (Cook's Theorem [4,5]). *Satisfiability of CNF formulae is an* **NP***-complete problem.* ∎

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

(3.4) Corollary *3SAT, satisfiability of CNFs all of whose clauses have 3 literals, is* **NP***-complete. (See [5].)* ∎

4 Spot scheduling

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

(4.1) Definition *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.

(4.2) Definition *Given an instance of a spot-scheduling problem, a* 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. The following lemma is easy to prove and useful.

(4.3) Lemma *If X is an* **NP***-complete problem, $Y \in$* **NP***, and $X \leq Y$ (see 2.8), then Y is* **NP***-complete.* ∎

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.

(4.4) Multi-partition.[2] 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 [5, section 4.2] is a restricted version where $n = 3m$ and by construction each sublist L_i contains 3 elements.

(4.5) Definition *Let X be a computation problem, and n a suitable measure of input size*[3] *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.*

(4.6) Proposition *(See [5].)* (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 polynomially bounded the problem is easy.* (iii) *The 3-partition problem is* **NP***-complete, even when the lengths B_i are small ($\leq 2^{16}n^4$).*

(4.7) Corollary *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 (2.5) to check that S satisfies the constraints (4.2). Therefore the problem is in **NP**.

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

[3] 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.

On the other hand, the 3-partition problem (4.4 is easily reduced 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. It is straightforward to show this is an easy reduction of 3-partition to spot-scheduling, which is therefore **NP**-complete. ∎

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

(4.8) Lemma *Let B be a fixed positive integer. Then the restricted form of multi-partition (4.4) where all breaklengths 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

$$x_1, \ldots, x_k$$

where (without loss of generality) $x_1 \geq x_2 \geq \cdots \geq x_k$ and $\sum x_j \leq B$. Let $p_1, \ldots, p_L$ be an enumeration of all these partitions.

Suppose that a bounded problem instance ℓ_j, $1 \leq j \leq n$; B_i, $1 \leq i \leq m$, is given. 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_k. By re-ordering the numbers B_i if necessary, it can be arranged that the partitions occur in blocks with non-decreasing indices:

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

Some of the blocks may be empty. 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 at most

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

such arrangements: a polynomial in m of degree $L-1$, which depends on B, the bound on breaklengths, 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 matches the lengths $\ell_1, \cdots, \ell_n$ and $B_1, \cdots, B_m$. This can be done in polynomial time. ∎

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

(4.9) Theorem *The related problem of spot-scheduling with colour restrictions is* **NP***-complete, even with bounded breaks.*

(4.10) 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 or 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. Formally, with the ith break 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 4.9

(5.1) Proof of Theorem 4.9 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 4.9 we introduce another **NP**-complete problem, a restricted form of 3SAT (3.4).

(5.2) Definition *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.

(5.3) Lemma *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 (2.5) to check that θ satisfies Φ, so Balanced 3SAT is in **NP**.

3SAT is known to be **NP**-complete (3.4), so by invoking Lemma 4.3 it is enough to show that

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

The ideas used here are those used to prove Lemma 3.4 [5].

Let Ψ be any 3CNF. We need to construct a balanced 3CNF Φ which is satisfiable if and only if Φ 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.[4] 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 and all these new variables V_j true satisfies all these new clauses, whether $\theta(L)$ is true (1) or false (0).

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

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

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

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

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 (2.5), so

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

as asserted. ∎

(5.4) In order to prove Theorem 4.9, it is enough to show that

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

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, with explanations following.

Φ	Σ
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 and $c_{\overline{X}}$, lengths 1 and 2, respectively
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 be called *selection spots.*

  A truth-assignment selects the truth-value 1 or 0 for X; the selection colours ensure 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. B 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 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.

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.

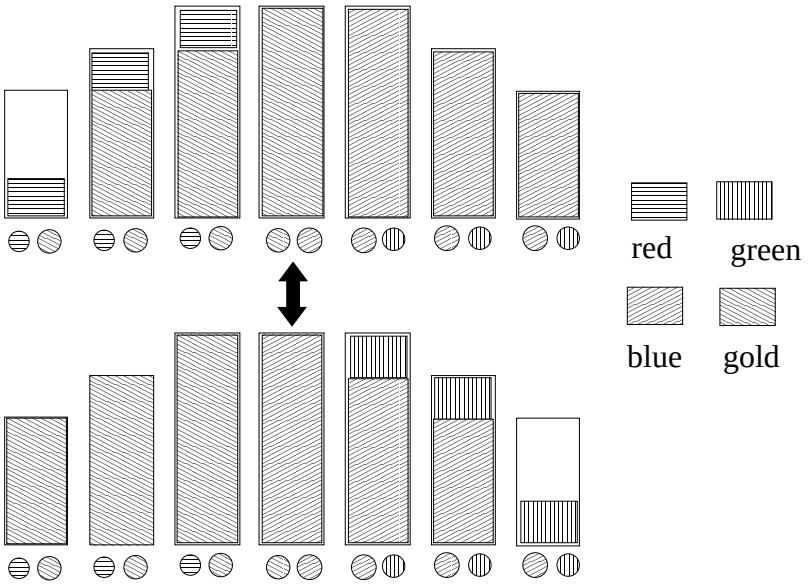

Figure 1: selection subsystem where $k = 3$, c_X is green, d_X gold, $c_{\overline{X}}$ red, and $d_{\overline{X}}$ blue. Breaks and spots are rectangular. The coloured circles below a break show which colours are allowed in that break.

(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 these breaks completely. As a result, no green spots can be scheduled in these breaks, but it is possible to schedule k double red spots. More formally,

(5.12) Lemma *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 $\ell = 2(k + j - 1)$. There are $k - j + 1$ selection spots of colour d_X and length $\geq \ell$, 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. Its length is $4k - 2$ 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. ∎

(5.13) Lemma *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.

- Note that for any literal L, unit c_L spots and double $c_{\overline{L}}$ spots in the clause-breaks means 'L true, $\overline{L}$ false' and vice-versa (see 5.9).

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

(5.14) Lemma *If Σ has a valid schedule then Φ is satisfiable.*

Proof. Let S be a valid schedule for Σ. We should be able to construct a truth-assignment θ on the assumption that S schedules unit (respectively, double) c_X spots in clause breaks means $\theta(X) = 1$ (respectively, $\theta(X) = 0$). However, there is nothing to prevent S from mapping both unit and double c_X spots to clause breaks.

For the last time, let X be a Boolean variable occurring k times in Φ (as does $\overline{X}$). There are $4k$ literal spots available for X, and $4k$ breaks in which they can be scheduled: $2k$ clause breaks and $2k$ selection breaks. Define

- $\theta(X) = 1$ if S maps some c_X spot into a selection break.

- $\theta(X) = 0$ if S maps some $c_{\overline{X}}$ spot into a selection break.

- By Lemma 5.12 (ii), these cases are mutually exclusive.

We define a related schedule S'.

Suppose, without loss of generality, that $\theta(X) = 1$. Then S assigns none of the literal $c_{\overline{X}}$ spots to selection breaks. They must all be mapped to clause breaks and surplus breaks. All

the unit $c_{\overline{X}}$ spots map to surplus breaks, since they have length 1. Therefore all the double $C_{\overline{X}}$ spots map to clause breaks.

The only way, therefore, in which the literal spots for X are mixed is where unit c_X spots are scheduled in selection breaks. But these spots can be swapped with double c_X spots scheduled in clause breaks. Since the selection breaks have room for double spots, the alterations do not violate the length constraints.

Thus we arrive at a valid schedule S' with the property that if $\theta(X) = 1$ then only unit c_X spots and double $c_{\overline{X}}$ spots are scheduled in clause breaks. Since every clause break now contains three literal spots of total length ≤ 5, every clause contains a literal true under θ: θ satisfies Φ, so Φ is satisfiable. ∎

(5.15) Proof of weak form of Theorem 4.9 (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 corresponding spot-scheduling problem (with colour restrictions) Σ which has a solution if (Lemma 5.13) and only if (Lemma 5.14) Φ is satisfiable. Σ is obviously easy to construct from Φ. Therefore

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

The former is **NP**-complete (5.3) and the latter is in **NP**, so it is **NP**-complete (4.3). ∎

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

(5.16) Lemma *A restricted form of BALANCED 3SAT, in which each literal occurs at most 3 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 $k > 3$ 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

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

where the X_i are $k-1$ new variables. (These clauses preserve balance, since every variable occurs as often as its complement.) Now each X_i and its complement occurs twice. and each A_i and its complement occurs 3 times.

Repeat this procedure for every variable occurring more than 3 times in Φ, obtaining a balanced 3CNF Ψ. It is easy to convert a satisfying truth-assignment for Φ to one for Ψ and vice-versa. Thus Φ is satisfiable iff Ψ is. Construction of Ψ from Φ is easy, and Ψ has the desired form (balanced 3CNF in which no literal occurs more than 3 times). Therefore the satisfiability problem for 3CNFs of the desired form is **NP**-complete. ∎

(5.17) Proof of Theorem 4.9. The proof of the weak form (5.15) argued that satisfiability of a Balanced 3CNF Φ is easily reduced to a spot-scheduling problem Σ.

By Lemma 5.16, we can restrict our attention to those balanced 3CNFs Φ in which every literal occurs at most 3 times.

With this assumption, referring to the table 5.5, the parameter k (number of occurrences of a literal) is at most 3. The selection breaks have length at most $4k-2$, i.e., at most 10. The other breaks have lengths 1 and 5. So the longest break is 10 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 [2]) means re-scheduling existing spots to make room for an incoming spot.

Spot dispersal works with or without colour restrictions and/or spot fixing (4.10).

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

(6.2) Dispersal chains (See [2]). Let S be a schedule solving a certain unit-length scheduling problem. A *dispersal chain* is a mixed sequence of breaks and 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 constrants (4.2, 4.10).

(6.3) Lemma *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 along 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: (sketch). Suppose that a full schedule S' exists for all the spots, including the last red one. Construct a directed graph G, nodes corresponding to breaks, with an edge connecting break b to break c, if S schedules a spot in b but S' schedules the same spot in c ($c \neq b$).

A dispersal chain corresponds to a simple path in G. Randomly try to build a simple path from R to N. If one succeeds without encountering a cycle, one has a dispersal chain. Otherwise, finding a simple cycle, use it to modify S', obtaining another full schedule which agrees more closely with S. So after finitely many steps one discovers a 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).

(6.5) Lemma *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 $i \leq j$.* ∎

Given a break b in B_i, constucting a dispersal chain from b to a break in B_0 is straightforward.

(6.6) Theorem *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 decreasing 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 this case, stop and report 'problem unsolvable.'

  Otherwise, re-order the breaks according to descending *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 efficiently — in linear time— using suitable linked structures. A runtime directly proportional to the input length is possible. Its *correctness* is not obvious.

(6.9) Lemma *If any schedule is possible, then this method will compute one.*

Proof. The proof is a kind of induction; that is, we imagine the method as proceeding as far as it can until it 'makes a wrong move.' So we can assume that it constructs a partial schedule S for some of the spots, and is unable to schedule the next spot, although spot dispersal would permit it to do so.

Suppose for clarity that the method fails to schedule a red spot.

Since spot dispersal is assumed possible, the red spot can be scheduled if we shift spots along a chain of breaks ending at a non-full break. Suppose for clarity that the last shift moves a yellow spot from a break B into a break C in N, where N denotes the set of breaks not yet full.

The off-line scheduling method is currently scheduling red spots and has finished scheduling all yellow spots. There may be several ways to apply spot-dispersal, but without loss of generality, yellow is the most recently scheduled colour occurring first (6.2) in some dispersal chain.

Consider the set A of all colours which have been scheduled after yellow, including red. The assumption about yellow implies that every colour in A is represented in every break in N. Therefore after the yellow spots were scheduled each break in N, including C, had more than $|A|$ units of free space.

The break B is full and contains no red spot, so the yellow spot was scheduled in B, when it had at most $|A|$ units of free space. This is impossible, since C should have been allocated a yellow spot before B was. ∎

7 References

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

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

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

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

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

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

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