OPERATIONS RESEARCH
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0030-364X | EISSN 1526-5463 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

Conflict resolution in the scheduling of television commercials

Daya Ram Gaur
University of Lethbridge, gaur@cs.uleth.ca

Ramesh Krishnamurti
Simon Fraser University, ramesh@cs.sfu.ca

Rajeev Kohli
Columbia University, rk35@columbia.edu

We extend a model by Bollapragada and Garbiras (2004) for scheduling commercial advertisements during breaks in television programming. The proposed extension allows differential weighting of conflicts between pairs of commercials. We formulate the problem as a capacitated generalization of the max k-cut problem in which the vertices of a graph correspond to commercial insertions and the edge weights to the conflicts between pairs of insertions. The objective is to partition the vertices into k capacitated sets so as to maximize the sum of conflict weights across partitions. We note that the problem is NP-Hard. We extend a local-search procedure due to Bollapragada and Garbiras (2004) to allow for the differential weighting of edge weights. We show that for problems with equal insertion lengths and break durations, the worst-case bound on the performance of the proposed algorithm increases with the number of program breaks and the number of insertions per break, and that it is independent of the number of conflicts between pairs of insertions. Simulation results suggest that the algorithm performs well even if the problem size is small.

Subject classifications: Marketing: advertising and media; networks/graphs: maxcut; heuristics.
Area of review: Marketing.

1. Introduction

The purpose of this paper is to propose a generalization of a model by Bollapragada and Garbiras (2004) for scheduling television commercials in program breaks. The generalization allows differential weighting of conflicts between pairs of insertions. This allows, for example, a distinction between multiple insertions of the same commercial, which are typically prohibited in the same program break, and insertions of competing commercials that are undesirable but possibly acceptable in the same program break. There can also be occasions when an advertiser stipulates a subset of certain competing brands that must be strictly excluded from placement in the same break. Such a condition can be accommodated by the proposed generalization. Finally, differences in conflict weights can be used to represent varying degrees to which competing commercials, or

classes of commercials, are in conflict. For example, it is likely less desirable that the same program break contain commercials of directly competing brands (e.g., BMW and Mercedes cars) than for brands that are more distant competitors (e.g., Mercedes cars and Ford trucks). The special case in which all conflict weights are zeroes or ones corresponds to the assumption of equal conflict weights by Bollapragada and Garbiras (2004).

We call the proposed problem the conflict-resolution problem in media scheduling. We discuss its relationship with the bin-packing and maxcut problems. We extend a local-search algorithm by Bollapragada and Garbiras (2004) to allow for differences in conflict weights. We show that the worst-case bound of the algorithm improves with the numbers of program breaks and commercials per break. Simulation results suggest that the proposed algorithm provides near-optimal solutions.

2. The conflict-resolution problem

The problem we examine is defined separately for each television show. This is because advertisers choose the programs in which they want to show the commercials for a product. The scheduling of commercials should not reassign these commercials to programs. A program break typically contains three to five slots, each of which corresponds to a position in the sequence in which commercials are aired within the break. A commercial assigned to a slot is called an *insertion*. We assume that all commercials are integer multiples of a minimum length. Any possible duration — for example, 1 second — can be used as a minimum length, although the typical durations of commercials are multiples of 15 seconds. We assume that each program break is an integer multiple of the minimum length, and that each program break has the capacity for the same number of minimum-length insertions. Note that our minimum length is different from the 30-second unit length used for pricing TV advertising.

Let k denote the number of program breaks. Let m denote the total number of insertions in a media schedule. Some insertions might be for the same commercial that must be shown several times over the course of the media schedule. We allow the possibility that it is not possible to air the same commercial more than once in a single program break (this constraint is not considered by Bollapragada and Garbiras 2004). In addition, buyers may specify competing brands for which advertisements should not be shown in the same program break. Some competing brands might be produced by the same firm. For example, Luvs and Pampers are brands of diapers that are both manufactured by Proctor and Gamble but that should typically not be advertised in the same program break. Other competing brands might be produced by different firms. For example, Acura, Lexus and Sonata are three competing cars manufactured by Honda, Lexus and Hyundai,

respectively. However, Acura and Lexus compete more closely against each other than they do against Sonata. Our model allows for differences in the conflict weights associated with pairs of insertions. The weights for some pairs of insertions can be so large that they never appear together in a break; e.g., insertions representing two instances of the same commercial, or two commercials for directly competing brands. More generally, it can be useful to have classes of conflicts, each class specifying the degree to which two commercials should not appear together in a break.

We associate a non-negative conflict weight with each pair of insertions. A zero weight means that the associated pair of insertions do not have a conflict. A larger weight, assigned separately to each pair of commercials, or to pairs of commercials in conflict classes, implies greater conflict. Consider, for example, assigning a weight of 1 to pairs of insertions in one class of competing commercials, a weight of 2 to insertions in another competing class, and a weight $w \gg 2$ to pairs of insertions corresponding to duplicates of the same commercial. This allows differential weighting of conflicts in the first two classes of commercials, and if w is large enough, ensures that commercials does not appear as duplicate insertions in the same program break.

Our solution procedure begins with a feasible assignment of insertions to program breaks. Finding such a feasible assignment is NP-Complete if the commercials have different, arbitrary lengths. To see this, consider program breaks of equal durations. Suppose there are no commercials for competing products. The k program breaks then correspond to the bins, and the n insertions to the items that have to be placed in the bins. Finding an assignment that packs all n items in the k bins is NP-Complete even for $k = 2$ (follows from PARTITION, Garey and Johnson 1979). If there are only l item sizes, and each item is at least of size $1/j$ of the bin capacity, then the bin packing problem can be solved in time linear in n, to which must be added a constant that is exponential in l, and doubly exponential in j (Blazewicz and Ecker, 1983). Both these constraints are likely to hold in practice for the problem of scheduling television commercials since most commercial breaks are an integer multiple of 60 seconds, and commercials of shorter duration (e.g., 15 or 30 seconds) can be combined to form 60 second "units". However, even if a feasible assignment is attained, the conflict-resolution problem is NP-Hard, because it is equivalent to a capacitated version of the maxcut problem.

2.1. Graph representation and formulation

Consider a graph $G(V, E)$ with $|V| = m$ vertices and $|E|$ edges. Each vertex represents an insertion and each edge connects a pair of conflicting insertions. We assign a conflict weight w_e to edge $e \in E$. A k-cut of the graph places the vertices into k mutually exclusive sets, $V_1, \ldots, V_k$, each

set corresponding to a distinct program break. Each break has a capacity of at least $n \geq m/k$ insertions, where $n \geq 1$ is an integer. If $kn > m$, then there is excess capacity. In this case, we add $kn - m$ dummy vertices, with no incident edges. Without loss of generality, we assume $m = kn$, and that each partition has a capacity of exactly $n = m/k$ vertices (insertions).

The conflict-resolution problem is to find an assignment of the insertions to the k program breaks so that the sum of the conflict weights across all pairs of program breaks is as large as possible. Let w_{uv} denote the conflict weight associated with insertions u and v, for all $u, v \in V$. Let x_{ui} denote a 0–1 integer variable, which takes a value of 1 only when insertion u is in program break i. Let y_{uivj} denote a $0-1$ integer variable that takes a value of 1 only if insertion u is in program break i and insertion v is in a different program break j. The conflict-resolution problem can be formulated as the following 0-1 integer program.

$$\text{Maximize} \sum_{(u,v)\in E} \sum_{i,j\in\{1,\ldots,k\}, i\neq j} w_{uv} y_{uivj}$$

Subject to :

$$y_{uivj} \leq \frac{1}{2}(x_{ui} + x_{vj}), \quad \text{for all } (u,v) \in E, \quad i < j, \; i,j \in \{1,\ldots,k\}$$
$$\sum_{i=1}^{k} x_{ui} \leq 1, \quad \text{for all } u \in V$$
$$\sum_{u\in V} l_u x_{ui} \leq n_i, \quad \text{for all } i \in \{1,\ldots,k\}$$
$$x_{ui} \in \{0,1\}, \quad \text{for all } u \in V, \quad i \in \{1,\ldots,k\}$$
$$y_{uivj} \in \{0,1\}, \quad \text{for all } (u,v) \in E, \quad i,j \in \{1,\ldots,k\}$$

The first constraint restricts the summing of conflict weights in the objective function to only those cases where the associated insertions are in different program breaks. The second constraint requires that each insertion should only be assigned to one program break. The third constraint specifies the capacity restriction for each program break: n_i denotes the number of units of the minimum unit length available in program break i, and l_u denotes the duration of insertion u as a multiple of the minimum unit length. Insertions and program breaks of different lengths can be allowed by modifying the third set of constraints in an obvious manner.

The conflict-resolution problem generalizes the max k-cut problem in which the vertices of a graph are assigned to subsets so that the sum of edge weights across all pairs of subsets is as large as possible. The generalization comprises (i) allowing different weights l_u to be associated with the vertices (insertions); and (ii) imposing constraints on the capacities of the subsets. The special case $k = 2$ with equal capacities and equal weighting of vertices is considered by Ageev and

Sviridenko (2001), who describe an algorithm that guarantees a solution value at least 1/2 of the optimal solution value. Ageev and Sviridenko (2000) consider a generalization of the capacitated max k-cut problem for hypergraphs. Their algorithm obtains an approximation ratio of 1/2 for the max k-cut problem with fixed, though possibly different, sizes of subsets. Both results use a pipage-rounding technique. Andersson (1999) considers equal capacities and describes an algorithm that obtains a $1-1/k+\Omega(1/k^3)$ performance guarantee. Feige and Langberg (2001) allow unequal capacities when $k=2$. They use semi-definite programming to construct an approximation algorithm with a lower bound of $1/2+\epsilon$, where $\epsilon>0$ is a fixed constant. Solution procedures for the max k-cut problem without capacity constraints include an algorithm by Frieze and Jerrum (1997) that extends a randomized-rounding procedure using semi-definite programming due to Goemans and Williamson's (1995). The algorithm obtains solutions with expected value no smaller than $\alpha_k \sim (2\log k/)k^2$. Goemans and Williamson (2004) obtain a performance guarantee of $\frac{7}{12}+\frac{3}{4\pi^2}\arccos^2(-1/4)\approx 0.83601$ when $k=3$; the same lower bound is also obtained by de Klerk, Pasechnik and Warners (2004). Kann, Khanna, Lagergren and Panconesi (1997) show that unless P=NP, the best possible performance ratio attainable by a heuristic for the max k-cut problem is $1-1/(34k)$. This result also applies to the present capacitated version of the problem.

3. Subset-swapping algorithm

As the conflict-resolution problem is NP-Hard, it is unlikely that efficient, polynomial-time algorithms exist for its solution. We describe a polynomial-time local-search procedure (Papadimitriou and Steiglitz 1982, Gaur and Krishnamurti 2005) that seeks to improve a given feasible solution by swapping subsets of $t \geq 1$ insertions across pairs of program breaks. The special case with 0-1 edge weights corresponds to the algorithm described by Bollapragada and Garbiras (2004). Our algorithm has an efficiently-searchable neighborhood; i.e., it is possible to improve a locally non-optimal solution, and to verify that a solution is locally optimal, in polynomial time (Orlin, Punnen and Schulz 2004). We evaluate the algorithm for the case where all program breaks have the same number of commercial units. The algorithm generalizes in an obvious manner to allow different insertion capacities for different program breaks. We show that for all values of $t \geq 1$, the subset swapping algorithm guarantees a solution whose value is no smaller than $1-(1/(k+\frac{k-1}{n-1}))$ of the optimal, where k is the number of program breaks and n is the minimum capacity of any break.

Let W denote the sum of conflict weights across all pairs of insertions. Observe that the value of W depends on the problem instance, not on the assignment of insertions to program breaks. The subset-swapping algorithm takes the following two inputs: (i) an initial, feasible assignment

of insertions to the k program breaks; and (ii) a value $t \geq 1$ for the number of insertions that are exchanged between pairs of breaks at each step of the algorithm. If desired, the algorithm can be run using different values of t, starting with $t = 1$, and using the final solution obtained with $t = a$ as the starting solution for the algorithm when $t = a + 1$. The largest possible value for t is $n - 1$, where n denotes the number of insertions per break.

We define a *t-set* to be a subset of t insertions in a program break. We use the term *self edge* to refer to a conflict between a pair of insertions within a t-set; and the term *cross edge* to a conflict between a pair of insertions across distinct t-sets. Suppose that at step r, the algorithm assigns a set V_i^r of commercials to program break i. Let W_i^r denote the sum of conflict weights across all pairs of insertions in the set $V_i^r,\ i = 1, \ldots, k$. Let

$$W_S^r = \sum_{i=1}^{k} W_i^r, \quad \text{and} \quad W_C^r = W - W_S^r,$$

where W_S^r is the sum of the within-break conflict weights, and W_C^r is the sum of conflict weights across all pairs of insertions that are in different program breaks. At step $r+1$, the algorithm swaps t-sets S_i^{r+1} and S_j^{r+1} in any two distinct program breaks i and j, provided the swap (i) satisfies the capacity constraints for each program break and (ii) improves the solution value; i.e., if the swap produces a solution value $W_S^{r+1} < W_S^r$. The algorithm stops if there is no pair of program breaks for which an improvement can be obtained in the solution value. The algorithm extends to arbitrary conflict weights the solution procedure described by Bollapragada and Garbiras (2004), who consider the situation in which the conflict weights are 0–1 values that reflect either the presence or the absence of conflicts between pairs insertions. We note that if the accepting criteria is changed slightly, to accept swaps that do not strictly increase the solution value (as for example in the level 2 swaps of Bollapragada and Garbiras 2004), then it is not clear if the algorithm terminates. In practice, it is useful to run the algorithm with different starting solutions and then select the best assignment across the runs for a given problem instance.

An iteration of the subset-swapping algorithm requires the evaluation of at most $\binom{n}{t} \times \binom{n}{t}$ possible subsets of t insertions in program breaks i, j; and all $\binom{k}{2}$ possible pairs of program breaks. Thus, in the worst case, a recursion step can require the evaluation of

$$\binom{n}{t}\binom{n}{t}\binom{k}{2}$$

possible subsets. Indeed, there is at least one step — the last step — that requires all these comparisons. It is clearly possible to change the algorithm so that it selects the best solution across all such comparisons at each step. However, as we show below, this is not necessary, for even

without it, the swapping algorithm gives solutions very close to the optimal for problems with a large number of program breaks. The performance guarantee we obtain below applies as well to this possible change in the implementation of the algorithm. The number of iterations is bounded by W, and the total running time is bounded by

$$\binom{n}{t}\binom{n}{t}\binom{k}{2}W.$$

This running time is not polynomial in the length of the input for arbitrary W. We prove a theorem in the next section that, together with the results of Orlin, Punnen and Schulz (2004), implies that a solution within $(1-\epsilon)$ of the local optima can be obtained for a given value of t in running time that is polynomial in the input instance size and $1/\epsilon$.

4. Theoretical analysis of the subset-swapping algorithm

The conflict-resolution problem with arbitrary insertion lengths and arbitrary capacities for program breaks appears to be far too difficult for theoretical analysis. Indeed, as noted earlier, even the problem of finding a feasible solution in the general case is a hard, combinatorial problem. However, it is possible to analyze the simpler problem where all insertions have the same duration, and all program breaks have the same capacity. We present these results below. In the next section, we examine the empirical performance of the algorithm when the insertions have different lengths and the program breaks have different durations.

We recall that W_S and W_C denote the respective sums of the conflict weights within and across program breaks, and that $W = W_S + W_C$ denotes the sum of all conflict weights across all pairs of insertions. We refer to W_S as the *self-edge weight* in the graph representation of the conflict-resolution problem. For a similar reason, we refer to W_C as the *cross-edge weight.* We obtain a tight lower bound on the value of the performance ratio $\phi = W_C/W_C^*$, where W_C^* denotes the optimal solution value for an instance of the conflict-resolution problem. Observe that $W_C^* \leq W = W_S + W_C$, because the best possible assignment is obtained when there are no conflicts at all between insertions assigned to a break. Thus,

$$\phi = \frac{W_C}{W_C^*} \geq \frac{W_C}{W_C + W_S} = \frac{1}{1 + \frac{W_S}{W_C}}.$$

Theorem 1 *The performance ratio of the subset-swapping algorithm has a lower bound*

$$\phi \geq 1 - \frac{1}{k + \frac{k-1}{n-1}}.$$

Proof. We denote by U_i the set of insertions assigned by the subset-swapping algorithm to program break i, for all $i = 1, \ldots, k$. Consider t-sets S_i and S_j, where $S_i \subset U_i$ and $S_j \subset U_j$ are t-sets in different program breaks, i and j. Let w_{uv} denote the weight of edge (uv). Let $X(S_i)$ denote the sum of conflict weights from the t-set S_i to the remaining insertions in the same program break. In other words, $X(S_i) = \sum_{u \in S_i, v \in U_i \setminus S_i} w_{uv}$. Let $Y(S_i, S_j)$ denote the sum of conflict weights from each insertion in the t-set S_i to the insertions in program break U_j which are not in the t-set S_j. In other words, $Y(S_i, S_j) = \sum_{u \in S_i, v \in U_j \setminus S_j} (u, v)$. We know that when the subset-swapping algorithm terminates, it is not possible to reduce the sum of within-break conflict weights. Thus,

$$X(S_i) + X(S_j) \leq Y(S_i, S_j) + Y(S_j, S_i).$$

We call this inequality a *basic inequality*. There is one basic inequality for each pair of subsets $S_i \subset U_i$ and $S_j \subset U_j$, $1 \leq i < j \leq k$. If we list all basic inequalities for all pairs of program breaks, and then sum the left hand sides together, and the right hand sides together, we obtain the expression

$$\sum_{i,j \ : \ 1 \leq i < j \leq k} \sum_{S_j \subset U_j, |S_i| = t} \sum_{S_i \subset U_i, |S_j| = t} X(S_i) + X(S_j) \leq \sum_{i,j \ : \ 1 \leq i < j \leq k} \sum_{S_j \subset U_j, |S_i| = t} \sum_{S_i \subset U_i, |S_j| = t} Y(S_i, S_j) + Y(S_j, S_i).$$

We call this the *master inequality*. Our objective is to write the master inequality, without explicitly enumerating all basic inequalities.

In order to do so, we will determine the number of times a self-edge (u, v) occurs in U_i across all basic inequalities involving a pair of t-sets. We call this number the "coefficient" of the weight associated with the edge (pair of insertions) (u, v) in the master inequality. Edge (u, v) occurs for every t-set $S_i \subset U_i$ when (i) u is in a t-set in U_i, but v is not in the t-set; or (ii) v is in a t-set in U_i but u is not in the t-set. In either case, one or the other of u or v is already in the set of insertions S_i and the other is in $U_i \setminus S_i$. The remaining $t - 1$ insertions in a t-set can be chosen from the remaining $n - 2$ insertions in U_i in $\binom{n-2}{t-1}$ ways. For each such t-set, the t insertions in a t-set $S_j \in U_j$ can be chosen in $\binom{n}{t}$ ways. Therefore, there are

$$2\binom{n-2}{t-1}\binom{n}{t}$$

cases in which the weight associated with the edge (u, v) occurs in the left hand sides of a basic inequality. Note that each edge is counted twice, once for u, v each. Now the edge (u, v) occurs as a self-edge when the algorithm evaluates swaps of t-sets in each of the $k - 1$ pairs of program breaks $(U_i, U_j),\ j \neq i$. Thus, the coefficient for edge (u, v) has the value

$$2\binom{n-2}{t-1}\binom{n}{t}(k-1).$$

As (u, v) refers to any arbitrary self-edge, the weight associated with every self-edge has the above coefficient in the master inequality obtained by summing across all the basic inequalities.

Next, we count the number of cross-edges between the program breaks, that is, the number of conflicting pairs of insertions across program breaks. We consider a typical cross-edge (u, w), where u is in program break i, which is assigned the set U_i of insertions, and w is in a different program break j, which is assigned the set U_j of insertions. Edge (u, w) occurs when (i) u is in a t-set but w is not in a t-set; or (ii) w is in a t-set but u is not in a t-set. The first event occurs in

$$\binom{n-1}{t-1}\binom{n-1}{t}$$

ways, because u is already part of the t-set, and so there are only $t-1$ insertions to choose from the remaining $n-1$ insertions in U_i. The coefficient of the weight associated with edge (u, w) in the master inequality is

$$2\binom{n-1}{t-1}\binom{n-1}{t}.$$

Once again, (u, w) can be any cross-edge, and so the above expression gives the coefficient of every cross-edge in the master inequality.

In summary, the weight associated with every self-edge on the left hand side of the master inequality has coefficient

$$2\binom{n-2}{t-1}\binom{n}{t}(k-1);$$

and the weight associated with every cross-edge on the right hand side of the master inequality has coefficient

$$2\binom{n-1}{t-1}\binom{n-1}{t}.$$

We now have sufficient information to write the master inequality:

$$2\binom{n-2}{t-1}\binom{n}{t}(k-1)W_S \leq 2\binom{n-1}{t-1}\binom{n-1}{t}W_C,$$

where W_S represents the sum of the weights of the self edges and W_C represents the sum of the weights of the cross edges. Thus,

$$\frac{W_S}{W_C} \leq \frac{\binom{n-1}{t-1}\binom{n-1}{t}}{\binom{n-2}{t-1}\binom{n}{t}(k-1)} = \frac{1}{k-1}\Big(1-\frac{1}{n}\Big).$$

The above upper bound for W_S/W_C implies

$$\phi \geq \frac{1}{1+\frac{W_S}{W_C}} \geq \frac{1}{1+\frac{1}{k-1}\Big(1-\frac{1}{n}\Big)} = \frac{k-1}{k-\frac{1}{n}} = 1-\frac{1}{k+\frac{k-1}{n-1}}.$$

□

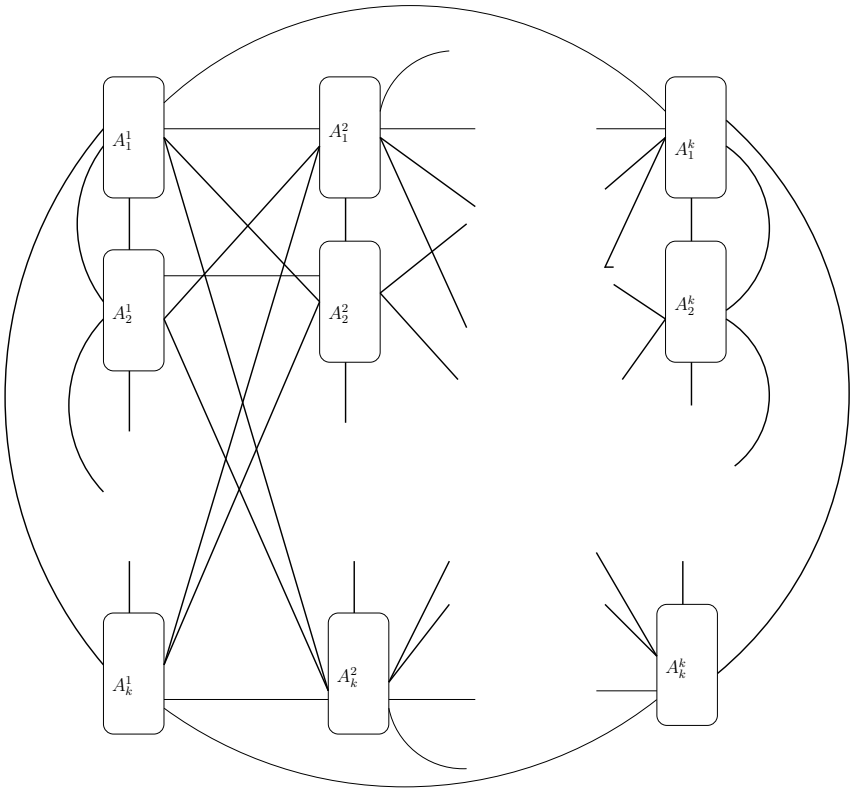

Figure 1 An example of a conflict-resolution problem for which the placement of insertions in program breaks attains the worst-case bound of Theorem 1.

Figure 1 shows an example for which the subset-swapping algorithm achieves the lower bound in Theorem 1. There are k program breaks and nk^2 insertions. The insertions are grouped into k^2 sets, $A_i^j, i, j = 1, \ldots, k$. Each set A_i^j contains n insertions. Consider a pair of sets, A_i^j and $A_i^s,\ j \neq s$, that appear in row i. There are n conflicts between the insertions in each such pair of sets, depicted using the "thin" lines in each row. Next, consider a pair of sets, A_i^j and $A_r^s, i \neq r$, that appear in different rows. Each insertion in A_i^j conflicts with each insertion in A_r^s, so that there are a total of n^2 conflicts across each pair of sets when $i \neq r$. We represent these n^2 conflicts by a "thick" line between the pairs of sets that do not appear in the same row in Figure 1. Suppose all insertions in a column — there are kn of these — appear in the same program break. Let u, v be two insertions in different program breaks. Consider a swap involving insertions u and v. The number of cross edges incident on u or v is $n(k-1)+1$; and the number of self-edges is $n(k-1)$. Thus, the subset-swapping algorithm will not swap any pair of insertions u and v that belongs to two different columns (program breaks). The corresponding solution value is given by

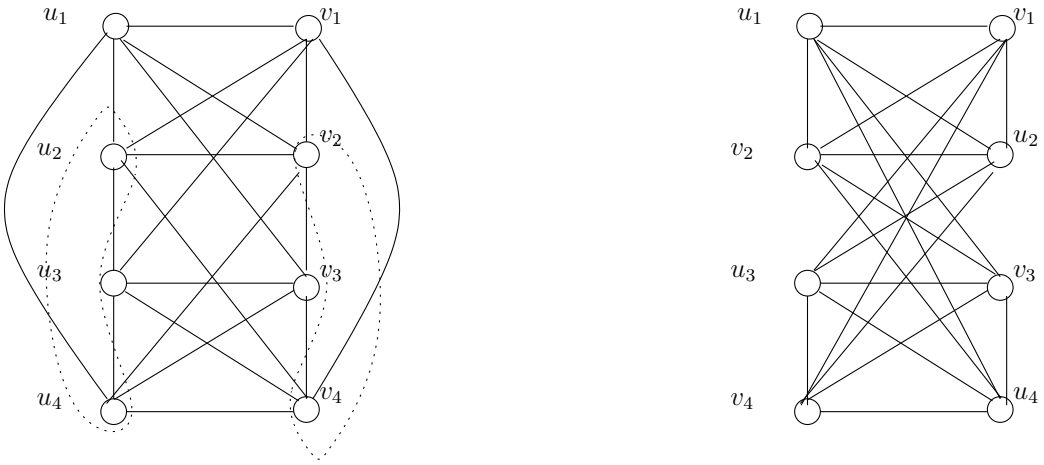

Figure 2 Example of a conflict-resolution problem for which a better assignment of insertions to breaks is obtained using subset size 2 than subset size 1.

$$W_C = \binom{k}{2} nk + \binom{k}{2} (k-1)n^2 k,$$

where the first term represents the sum of conflicts across subsets in the same row of Figure 1, and the second term represents the sum of conflicts across subsets in different rows and different columns. The optimal solution places all kn insertions in a row within the same program break. The corresponding solution value is given by

$$W_C^* = \binom{k}{2} n^2 k + \binom{k}{2} (k-1)n^2 k,$$

where the first term corresponds to the number of edges between sets in a column, and the second term corresponds to the number of edges between each pair of sets that lie in a different row and a different column. Substituting for W_C and W_C^* and simplifying the expression for the performance ratio gives

$$\phi = \frac{W_C}{W_C^*} = 1 - \frac{1}{k + \frac{k}{n-1}},$$

which approaches, with increasing values of k, the bound obtained in Theorem 1.

Although the lower bound in Theorem 1 is independent of the number of insertions swapped at each step of the algorithm, there can be instances of the conflict-resolution problem where a larger value of t obtains a better solution value. Figure 2 illustrates such a situation. The eight vertices in the left hand panel corresponds to commercial insertions. There are two program breaks. Insertions

u_1, u_2, u_3, u_4, are in one program break; and insertions v_1, v_2, v_3, v_4, are in the other program break. Suppose each conflict weight has a unit value. Then the assignment in the left hand panel has a solution value of 12, because it resolves the 12 conflicts between each u_i-v_j pair of insertions. It also has 8 unresolved conflicts, four in each program break. There is no possible exchange of $t = 1$ insertions across breaks that can improve the solution value. However, a better solution can be obtained by exchanging $t = 2$ insertions across program breaks. For example, exchanging the pair of insertions u_2 and u_4, with insertions v_2 and v_4, produces the assignment shown in the right hand panel of Figure 2. The total number of conflicts (and the sum of conflict weights) across program breaks increases from 12 to 16. Each program break now has two conflicting insertions: u_1-v_2 and u_3-v_4 in one program break; v_1-u_2 and v_3-u_4 in the other program break.

The lower bound in Theorem 1 strictly exceeds $1 - \frac{1}{k}$. If the number of program breaks is large enough, the subset-swapping algorithm will give solutions very close to the optimal, regardless of the maximum number of insertions in a break. A typical half hour show has three program breaks. A weekly half-hour show has $k = 12$ program breaks over a four week period, and so the performance ratio of the subset switching algorithm is no smaller than $11/12 = 0.917$. Programs like *Evening News, Oprah, Jeopardy* and *Wheel of Fortune,* as well as syndicated re-runs, have at least five airings a week, for a total of 15 breaks per week per show. Over a four-week planning period, these shows have 60 program breaks. The subset-switching algorithm then has a performance ratio no smaller than $59/60 = 0.983$. In practice, one needs to implement the algorithm across all shows, both for regular scheduling, and for the re-scheduling that becomes necessary when there are disruptions in scheduled programming. The latter occurs, for example, when there is breaking news, when live broadcasts of sporting events and awards ceremonies run longer or shorter than expected, and when new commercials are received for immediate scheduling by a television network.

5. Empirical testing

We generate 1800 problem instances, 100 per cell of a 2×9 experimental plan in which the design factors are the number of program breaks ($k = 2, \ldots, 10$) and the insertion lengths (all equal lengths, or long and short insertions). A short insertion has a length of one unit (e.g., 30 seconds), and a long insertion has a length of two units. We constrain each program break to have 5 insertions, four of unit lengths, and a fifth of one or two unit lengths. A problem with k program breaks thus has $5k$ insertions and no excess capacity, which can only improve the performance of the algorithm.

We start with a complete k-partite graph with 5 vertices in each subset and generate each edge weight by sampling from a uniform distribution in [0, 1]. To generate insertions that must appear

in different program breaks, we change the weight on each edge to 100 with probability $p = 0.05$. As all the edges are across partitions in a k-partite graph, the weight of the optimal solution equals the sum of the weights of all the edges. We obtain an isomorphic image of the graph by randomly permuting the labels on the vertices and using it as an initial solution for the subset-swapping algorithm. We implement the subset-swapping algorithm with $t = 1$ (the solution can only improve if we also consider larger values of t). We restrict the feasible swaps to either a pair of insertions with the same weight, or to an insertion with weight 2 in one subset and a pair of insertions with unit weights in another subset. We solve each problem 50 times using different starting solutions, and retain the best solution as the final solution for the problem. We implement the algorithm in Python on a 2.80 GHz Intel Pentium 4 with 1 GB of RAM. The average running time across the 50 random restarts is 1.47 minutes per problem instance.

In all 1800 problems, the subset-swapping algorithm assigns all insertions that must be strictly separated to different program breaks. Figure 3 shows the average value of the empirical performance ratio for the subset-swapping algorithm for equal and unequal insertion lengths, and for each value of $k = 2, \ldots, 10$. Varying the length of insertions has virtually no effect on the empirical performance ratio, which in all cases remains substantially better than the worst-case bound. For example, the average value of the performance ratio is 0.94 when $k = 2$; the worst-case bound is 0.5556. The results suggest that the subset-swapping algorithm runs quickly and performs well, even for small problems, when the conflicts have different weights. Bollapragada and Garbiras (2004) obtain similar results for problems with $t = 1$ and 0-1 conflict weights.

6. Conclusion

We describe a method to assign commercial insertions to breaks in television programming. The method extends a model by Bollapragada and Garbiras (2004) by allowing the conflict weights to represent the degrees to which pairs (or classes) of commercials are in conflict. Our method uses a capacitated max k-cut representation of the problem. Insertions are represented by the vertices of a graph, and conflicts between pairs of insertions by edge weights. We partition the vertices into k sets using a subset-swapping algorithm that seeks to maximize the sum of edge weights across subsets. The proposed algorithm in an extension of the local-search heuristic described by Bollapragada and Garbiras (2004) that allows differences in conflict weights. We show that the algorithm guarantees a solution within at least $1 - (1/(k + \frac{k-1}{n-1}))$ of the optimal when there are k program breaks. This guarantee is large enough for large k; a simulation suggests that the algorithm performs well for small values of k as well.

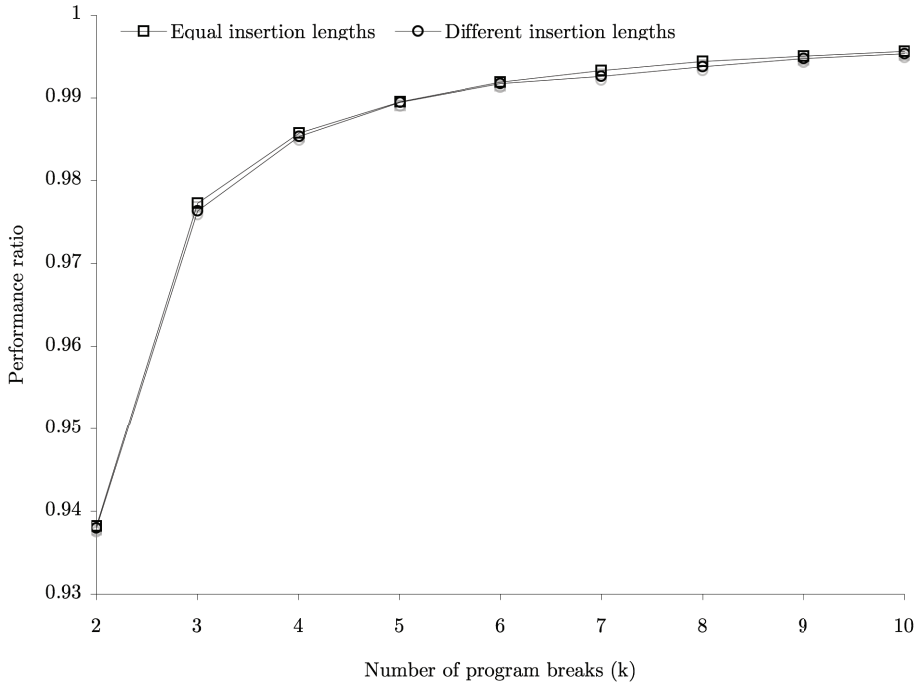

Figure 3 Empirical performance of the subset-swapping algorithm.

The maxcut problem, with and without the capacity constraint, is relevant to several other marketing problems. An example is the partitioning of such items as books sold by a retailer into k sets in such a way that items in the same (different) set are purchased most (least) often with each other. Another example is the scheduling of television shows in day parts to maximize the total ratings over a planning period. This problem can be formulated as a capacitated max k-cut problem if lead-in programs have little effect on a show's ratings, and if the positions of shows on competing channels are unchanged (see, e.g., Reddy, Aronson and Stam 1998). A final example, which does not require capacity constraints, concerns market segmentation. In this case, the vertices of the graph represent individuals and the edge weights represent a measure of dissimilarity between pairs of individuals. The objective is to group the individuals into k segments so that an aggregate measure of dissimilarity across segments is as large as possible.

Acknowledgements: The first two authors are grateful to the Natural Sciences and Engineering Research Council of Canada for providing research support.

References

[1]Ageev, A. A. and M. Sviridenko, "An approximation algorithm for hypergraph max k-cut with given sizes of parts," *Lecture Notes in Computer Science (Proceedings of ESA '00),* **1879** (2000), 32–41.

[2]Ageev, A. A. and M. Sviridenko, "A 0.5-approximation algorithm max dicut with given sizes of parts," *SIAM Journal on Discrete Mathematics,* **14(2)** (2001), 246–255.

[3]Andersson, G., "An approximation algorithm for max p-section," *Lecture Notes in Computer Science (Proceedings of STACS'99),* **1563** (1999), 237–247.

[4]Bollapragada, S. and M. Garbiras, "Scheduling commercials on broadcast television," *Operation Research* **52 (3)** (2004), 337–345.

[5]J. Blazewicz and K. Ecker, "A linear time algorithm for restricted bin packing and scheduling problems," *Operations Research Letters* **2** (1983), 80–83.

[6]de Klerk, E., D. V. Pasechnik, and J. P. Warners, "On approximate graph colouring and max-k-cut algorithms based on the theta-function," *Journal of Combinatorial Optimization,* **8** (3), (2004), 267–294.

[7]Frieze, A. and M. Jerrum, "Improved approximation algorithms for max k-cut and max bisection," *Algorithmica* **18 (1)** (1997), 67–81.

[8]Garey, M.R. and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness,* Freeman, San Francisco, 1979.

[9]Gaur, D. and R. Krishnamurti, "The capacitated max-k-cut problem," *Proceedings of the International Conference on Computational Science and its Applications,* (2005), LNCS 3483, 670–679.

[10]Goemans, M.X. and D.P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the Association of Computing Machinery,* **42** (1995), 1115–1145.

[11]Goemans, M.X. and D.P. Williamson, "Approximation algorithms for max-3-cut and other problems via complex semidefinite programming," *Journal of Compuer and System Sciences ,* **68 (2)** (2004), 442–470.

[12]Kann, V., S. Khanna, J. Lagergren, and A. Panconesi, "On the hardness of approximating max k-cut and its dual," *Chicago Journal of Theoretical Computer Science,* **1997**, (1997).

[13]Mahajan, S. and H. Ramesh, "Derandomizing approximation algorithms based on semidefinite programming," *SIAM Journal on Computing,* **28** (1999), 164–1663.

[14]Orlin, J. B., A. P. Punnen and A. S. Schulz, "Approximate local search in combinatorial optimization," *SIAM Journal on Computing,* **33 (5)** (2004), 1201–1214.

[15]Papadimitriou, C. H. and K. Steiglitz, *Combinatorial optimization: algorithms and complexity,* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[16]Reddy, S.K., J.E. Aronson and A. Stam, "SPOT: scheduling programs optimally for television," *Management Science,* **44 (1)**, (1998), 83–102.