Finding optimal realignments in sports leagues using a branch-and-cut-and-price approach

Xiaoyun Ji and
John E. Mitchell*

Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Fax:(518)276-4824 Email: jix@rpi.edu
E-mail: mitchj@rpi.edu
*Corresponding author

Abstract: The sports team realignment problem can be modelled as k-way equipartition: given a complete graph $K_n = (V, E)$, with edge weight c_e on each edge, partition the vertices V into k divisions that have exactly S vertices, so as to minimize the total weight of the edges that have both endpoints in the same division. In this paper, we discuss solving k-way equipartition problem using branch-and-price scheme. We demonstrated the necessity of cutting planes for this problem and suggested an effective way of adding cutting planes in the branch-and-price framework. We solved the pricing subproblem as an integer programming problem. Using this method, we found the optimal realignment solution for three major professional sports leagues in North America (basketball, hockey, football). We also present computational results on some larger randomly generated micro-aggregation problems.

Keywords: Realignment, graph equipartition, branch-and-price, clustering, micro-aggregation

Reference to this paper should be made as follows: Ji, X. and Mitchell, J. E. (2005) 'Finding optimal realignments in sports leagues using a branch-and-cut-and-price approach', *Int. J. Operational Research*, Vol. 1, No. 1, pp.xxx – xxx.

Biographical notes: Xiaoyun Ji received her PhD in Mathematics at Rensselaer Polytechnic Institute. She is a Postdoctoral Research Associate at the department of Mathematical Sciences, Rensselaer Polytechnic Institute. Her research interests include Integer Programming, Combinatorial Optimization Problems, Mathematical Programming with Equilibrium Constraints.
John Mitchell received his PhD in Operations Research and Industrial Engineering from Cornell University. He is a Professor of Mathematical Sciences at Rensselaer Polytechnic Institute. His research interests are generally in optimization, with emphases on interior point methods, integer programming, and the application of interior point methods to solve integer programming problems.
This material is based upon work supported by the National Science Foundation under Grant No. DMS–0317323. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National

Science Foundation.

1 Introduction

In sports team realignment problems, a set of teams is divided into divisions of relatively equal size. Every team plays against each other team in its own division, and then they play against some opponents from outside divisions depending on previous results. For example, the Nation Basketball Association (NBA) in United States realigned their 30 teams for 2004-05 season into eastern and western conference, see NBA (2003), each conference has a total of 15 teams with 3 divisions of 5 teams. Teams will play divisional opponents four times each, conference opponents outside the division three or four times each and opponents outside the conference two times each.

To model this realignment problem, we regard the NBA teams as the vertices of a graph. The distance between every pair of teams is the cost for the edge connecting the corresponding vertices. Any group of 5 teams is a valid division. A partition of the 30 teams into 6 divisions each containing 5 teams is a valid alignment. The total intradivisional travel distance is the sum of the edge costs of all the edges that have both endpoints in the same division. There are many ways to arrange the divisions. Since the choice of outside division opponents varies from season to season, and the majority of the games are in-division games, we choose the divisions to minimize the total intradivisional travel distances. This objective will place nearby cities in the same division, creating natural rivalries.

The optimal solution according to this objective is shown in Figures 3 - 4 for NBA, National Hockey League (NHL) and National Football League (NFL), with details in Tables 1 - 3.

When it is possible to divide the teams into divisions of equal size, the problem can be modelled as k-way equipartition problems. Given an undirected graph $G(V, E)$, with edge cost $c_e, e \in E$, the k-way equipartition problem is to divide the vertices V into k sets of vertices $\Pi = \{P_1, P_2, ..., P_k\}$, each of the same size, so as to minimize the total cost of the edges which have both endpoints in one of the sets. This is equivalent to the problem of dividing V into clusters of size $S := |V|/k$. Notice this is possible only when $|V|$ is a multiple of S, which we will assume in this paper. When $|V|$ is not a multiple of S, the realignment problem can be modelled as a Clique Partition Problem with minimum size requirement (CPPMIN), where requires each division to have no less than S teams. A detailed discussion on CPPMIN can be found in Ji (2004).

The k-way equipartition problem is a special case of the following generalized k-way equipartition problem: with an additional node weight a_v given for each $v \in V$, the generalized k-way equipartition problem is to find a partition $\Pi = \{P_1, P_2, ..., P_k\}$ of V such that it solves

$$\begin{aligned} &\min \quad \sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ (1) \qquad &\text{s.t.} \quad \sum_{v \in P_i} a_v = S \qquad i = 1..k \end{aligned}$$

Here $E(P_i)$ is the set of edges connecting any two vertices in P_i. When the node weight a_v is fixed at 1, we have our k-way equipartition problem, In this paper, we only discuss k-way equipartition problem, but the analysis for our branch-and-price algorithm can be readily applied to generalized k-way partition problem. In fact, in our branch-and-price method, we will be actually solving a generalized k-way equipartition problem after branching happens.

Mitchell (2003) used the same k-way equipartition model for the NFL team realignment problem, and solved it as an integer programming problem using branch-and-cut method, see more details in Mitchell (2001). In this paper, we are going to solve the same problem, but using branch-and-price method. Other application of k-way equipartition includes micro-aggregation for statistical disclosure control (Domingo-Ferrer & Mateo-Sanz 2002), telecommunications (Lisser & Rendl 2003), communication centers locating problem (Guttmann-Beck & Rubinstein 1998).

Similar to the branch-and-cut method, the branch-and-price scheme for solving an integer programming problem consists of two major steps. The first is to solve the LP relaxation, the second is to use the LP solution to achieve optimal solution through branching. The major difference lies in the first stage on how to solve the LP relaxation. Branch-and-price solves the LP relaxation using column generation while branch-and-cut solves it using row generation. This difference is a consequence of the different IP formulations of the original problem. The IP problem formulated for branch-and-price usually has a large number of variables. Most of these variables and corresponding columns are left out of the LP relaxation initially because there are too many columns to handle efficiently and most of them will will not be used in an optimal solution anyway. To check the optimality of a current solution, a subproblem, called the pricing problem, is solved to try to identify columns to enter the basis. If such columns are found, it is added into the original formulation and the LP is re-optimized. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality condition.

Branch-and-price has been successfully used to solve a group of integer programming problems. A good introduction can be found in Barnhart et al. (1998). Wilhelm (2001) gave a review on applying column generation methods to solve IP problems with emphasis on formulation issues. Lübbecke & Desrosiers (2002) surveyed column generation with emphasis on the dual point of view. Savelsbergh (1997) discussed the branch-and-price algorithm on the generalized assignment problem. Other applications include crew scheduling (Vance et al. 1997), bin packing and cutting stock problems (Vanderbeck 1999), edge coloring problems (Nemhauser & Park 1991), graph coloring problems (Mehrotra & Trick 1996).

If we change the minimization in problem (1) into maximization and relax the size constraint to an upper bound, we get the min-cut clustering problem (see Johnson et al. 1993), also called clustering problem with knapsack constraints (see Mehrotra & Trick 1998):

$$\text{(2)} \qquad \begin{array}{ll} \max & \displaystyle\sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ \text{s.t.} & \displaystyle\sum_{v \in P_i} a_v \le S \qquad i = 1..k \end{array}$$

Johnson et al. (1993) considered this problem with k and S both given. They solved it using branch-and-price, with the column generation subproblem solved as an integer programming problem on the boolean quadratic polytope. They discussed some strong valid inequalities for their column generation subproblem and described their solution strategy with computational results. Mehrotra & Trick (1998) improved upon the results in Johnson et al. (1993) by using a combinatorial method to solve the column generation subproblem. Mehrotra et al. (1998) used the min-cut clustering problem to model the political redistricting problem. The knapsack constraint (2) is used to balance the population of each district; the objective function is set up to enforce compactness of each district.

Their successful application of column generation on this problem inspired us to try branch-and-price on the k-way equipartition problem. We will give our integer programming formulation in section 2. The two major differences between our branch-and-price method for k-way partition and the method used by Mehrotra & Trick (1998) for the clustering problem with knapsack constraints comes from the two differences between the original problems. First, the constant node weight $a_v = 1$ simplified our problem, so we are able to add in cutting planes to tighten up the LP relaxation. This is discussed in section 3. The change in the objective function changed the property of the column generation subproblem. The combinatorial method used by Mehrotra & Trick (1998) can not be applied in k-way equipartition. So we solved the column generation problem as an integer programming problem. This will be discussed in detail in section 4. The rest of the paper consists of the branching rule in section 5, the discussion on stabilization in section 6, an outline of the algorithm in section 7. Finally in section 8, we give the computational results. We give the optimal realignment solution for the basketball, football and hockey sports leagues in North America. We also test the performance of our algorithm on larger problems that are typical in statistical micro-aggregation problems.

2 An Integer Programming Formulation for Column Generation

Now we give the integer programming formulation of the k-way equipartition problem in a form that is suitable for column generation. Consider k-way equipartition on graph $G = (V, E)$ with each partition size $S = n/k$, a cluster $P \subseteq V$ is feasible if there are exactly S vertices in this cluster, i.e. $|P| = S$. For each feasible cluster P, we define a binary variable x_P

$$x_P = \begin{cases} 1 & \text{if cluster } P \text{ is used in the solution of } k\text{-way equipartition} \\ 0 & \text{otherwise} \end{cases}$$

Let $w_P = \sum_{e \in E(P)} w_e$, then the k-way equipartition problem can be formulated as the following IP problem, denoted as (MIP),

$$\begin{aligned} \min \quad & \sum_P w_P x_P \\ \text{s.t.} \quad & \sum_{P: v \in P} x_P = 1 \quad \forall v \in V \\ & x_P \in \{0, 1\} \\ & P \in 2^V, |P| = S \end{aligned}$$

Here $2^V = \{P : P \subseteq V\}$ represents the set of all subsets of V. The first constraint says that every node v must be covered in exactly one of the chosen clusters. In this paper, we are going to consider only nonnegative edge weights, so we automatically have $x_P \leq 1$. We then relax the integrality constraint, to get the linear relaxation (MLP):

$$\begin{aligned} \min \quad & \sum_P w_P x_P \\ \text{s.t.} \quad & \sum_{P: v \in P} x_P = 1 \quad \forall v \in V && (3) \\ & x_P \geq 0 && (4) \\ & P \in 2^V, |P| = S && (5) \end{aligned}$$

There are $\binom{n}{S}$ feasible clusters P that satisfy (5), thus $\binom{n}{S}$ variables x_P. But since most of them will be 0 in the optimal solution, we do not need to include all of them in the initial formulation. Instead, we can start with a subset of clusters, then add in the necessary ones later by solving a pricing algorithm. This step is called column generation, suitable for LP's with large number of variables. Starting with a subset of feasible clusters $T \subseteq 2^V$, we solve a restricted (MLP), called (RMLP), where $P \in T$. The optimal solution of (RMLP) is a feasible solution to MLP. The dual values π_v for each constraint in (RMLP) are used to decide whether we need to expand T.

We demonstrate how to make this decision by a simple example. In a k-way equipartition to divide a graph of 4 nodes into clusters of size 2, suppose we have considered 4 clusters, $P_1 = \{v_1, v_2\}$, $P_2 = \{v_3, v_4\}$, $P_3 = \{v_1, v_3\}$, $P_4 = \{v_2, v_4\}$, i.e., $T = \{P_1, P_2, P_3, P_4\}$. Now we consider if we need to expand T into $T' = \{P, P_2, P_3, P_4, P_5, P_6\}$, where $P_5 = \{v_1, v_4\}$, $P_6 = \{v_2, v_3\}$. The LP relaxation of (MIP) on T' can be written as the following:

$$\begin{array}{lccccccccc} \min & w_1x_1 & +w_2x_2 & +w_3x_3 & +w_4x_4 & +w_5x_5 & +w_6x_6 & & \\ \text{s.t.} & x_1 & & +x_3 & & +x_5 & & = & 1 \\ & x_1 & & & +x_4 & & +x_6 & = & 1 \\ & & x_2 & +x_3 & & & +x_6 & = & 1 \\ & & x_2 & & +x_4 & +x_5 & & = & 1 \\ & x_i & & & & & & \geq & 0 \end{array} \qquad (6)$$

Take π_v, α_v to be the dual variables corresponding to the equality and inequality constraints respectively. The dual problem is

$$\begin{array}{lcccccccccccc} \max & \pi_1 & +\pi_2 & +\pi_3 & +\pi_4 & & & & & & & & \\ \text{s.t.} & \pi_1 & +\pi_2 & & & +\alpha_1 & & & & & & = & w_1 \\ & & & \pi_3 & +\pi_4 & & +\alpha_2 & & & & & = & w_2 \\ & \pi_1 & & +\pi_3 & & & & +\alpha_3 & & & & = & w_3 \\ & & \pi_2 & & +\pi_4 & & & & +\alpha_4 & & & = & w_4 \\ & \pi_1 & & & +\pi_4 & & & & & +\alpha_5 & & = & w_5 \\ & & \pi_2 & +\pi_3 & & & & & & & +\alpha_6 & = & w_6 \\ & & & & & \alpha_i & & & & & & \geq & 0 \end{array} \qquad (7)$$

Let $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4)$ and $(\bar{\pi}, \bar{\alpha})$ be the optimal solution to the primal and dual pair, when x_5, x_6 were not introduced. $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, 0, 0)$ is also a feasible solution to (6). Only when the reduced cost of x_5, $w_5 - (\bar{\pi}_1 + \bar{\pi}_4)$, is negative, do we need to add in cluster P_5. Similarly, only when $w_6 - (\bar{\pi}_2 + \bar{\pi}_3)$ is negative, do we need to add in cluster P_6.

The above example demonstrates whether we need to expand T can be determined by solving a minimum node-edge-weighted clique problem (MINNEWCP) with minimum size constraints. This is our pricing problem. Given a graph $G = (V, E)$ with node weight $-\pi_v$ and edge weight w_{ij}, find a feasible cluster whose total weight (edge weights and node weights all together) is minimized. If the optimal value is non-negative, then there exist no improving clusters. Otherwise, any feasible cluster with a negative objective value provides an improving cluster.

This process should be repeated until there are no improving clusters. If the optimal solution to the final linear relaxation, (MLP), is an integer solution, then we are done, otherwise we need to use either cutting plane method or branching to force integrality.

3 Cutting Planes

We discuss the tightness of the LP relaxation (MLP) in this section. In the next section, we will give the details on how to solve the pricing problem.

The following example shows us that the (MLP) given before is not a very tight relaxation for (MIP). In Figure 1, the 12 vertices of a complete graph K_{12} is to be divided into $k = 3$ clusters of size $S = 4$. The edge costs are given as the following: $c_{ij} = 1$ for $1 \leq i < j \leq 6$, $c_{ij} = 1$ for $7 \leq i < j \leq 12$ and $c_{ij} = 10$ for all the edges $1 \leq i \leq 6 < 7 \leq j \leq 12$. For illustration purpose, only 8 edges out of the total $6 \times 6 = 36$ edges connecting vertices $\{1, 2, 3, 4, 5, 6\}$ to vertices$\{7, 8, , 9, 10, 11, 12\}$ are shown, but the missing ones also have edge weights of 10. It is easy to see that the optimal solution is to divide these 12 vertices into 3 clusters, for example, $P_1 = \{1, 2, 3, 4\}$, $P_2 = \{7, 8, 9, 10\}$, and $P_2 = \{5, 6, 11, 12\}$ with corresponding cluster weight $w_1 = 6$, $w_2 = 6$, $w_3 = 42$, giving the optimal objective value of $obj = 54$. But a fractional solution with $x_p = 0.5$, $p = 1..6$, where the 6 clusters are $P_1 = \{1, 2, 3, 4\}$, $P_2 = \{3, 4, 5, 6\}$, $P_3 = \{1, 2, 5, 6\}$, $P_4 = \{7, 8, 9, 10\}$, $P_5 = \{9, 10, 11, 12\}$, $P_6 = \{7, 8, 11, 12\}$, gives a much better objective value of $obj = 0.5 \times 6 \times 6 = 18$.

However, we can easily cut off this this fractional LP solution by using the following constraint $\sum_{P \subset \{1..6\}} x_P \leq 1 = \lfloor 6/4 \rfloor$, which is implied by the requirement that each cluster has to be of size $S = 4$ in this example.

Further generalizing the above example, we reach theorem 3.1.

Theorem 3.1. *Given $Q \subset V, |Q| < qS$, inequality* (8) *is a valid constraint for (MIP) on Q.*

$$\sum_{P : P \subseteq Q} x_P \leq q - 1 \qquad (8)$$

The proof is straight forward since when $|Q| < qS$, we can partition Q into at at most $q - 1$ clusters of size bigger or equal to S.

This constraint is similar to the pigeon constraint for the CPPMIN in Ji (2004). One difference we would like to mention here is that unlike the CPPMIN, when $Q = V(8)$ will be automatically satisfied by equipartition. In fact, it can be derived from the sum of all the equations of type (3).

With these new cutting planes, we write out our restricted (MLP) as (RMLP),

$$\begin{array}{llr}
\min & \sum_{P} w_P x_P & \\
\text{s.t.} & \sum_{P:v\in P} x_P = 1 \qquad \forall v \in V & (9) \\
& \sum_{P\subseteq Q_i} x_P \le \lfloor |Q_i|/S \rfloor \qquad i = 1..q & (10) \\
& x_P \ge 0 & (11) \\
& P \in T \subseteq 2^V, |P| \ge S & (12)
\end{array}$$

Let π_v, σ_i, and α_P be the dual variables corresponding to equations (9) - (11). We can write down the corresponding dual problem:

$$\begin{array}{llllllll}
\max & \sum_{v=1}^{n} \pi_v & + \sum_{i=1}^{q} \lfloor \frac{|Q_i|}{S} \rfloor \sigma_i & & & & & \\
\text{s.t.} & \sum_{v\in P} \pi_v & + \sum_{i:P\in Q_i} \sigma_i & +\alpha_P & = & w_P & \text{for every } P \in T & \\
& & \sigma_i & & \le 0 & & i = 1..q & (13) \\
& & & \alpha_P & \ge 0 & & \text{for every } P \in T &
\end{array}$$

We look for violated constraints by checking on the subsets of vertices generated by combining the columns corresponding to the non-zero variables in the optimal LP solution for the current (MLP) problem. Since every feasible cluster is of size S, the union of any two is between size $S + 1$ and $2S$. It turns out that only a small number of additional cutting planes leads to a much better approximation for MIP. In other words, the total number of cutting planes is not big, and each cutting plane involves no more than 2S vertices. This makes it possible for us to combine the cutting plane method into the column generation framework without complicating the pricing subproblem too much, which is usually a difficult task in general branch-and-price. Interested readers please refer to Ji & Mitchell (2005) for details on the cutting plane generation method and how they are incorporated into the branch-and-price framework.

4 Generate Improving Clusters

Finding a fast and good algorithm to solve the pricing problem, the Minimum Node-Edge-Weighted Clique Problem (MINNEWCP) with size constraints, is an essential part of the branch-and-price scheme. In this section, we will briefly explain the difficulty and our approach. For more details on a very similar pricing problem, interested reader can refer to Ji & Mitchell (2005).

4.1 Problem Definition

We state the problem formally as the following. Given a graph $G = (V, E)$, a weight π_v associated with each vertex $v \in V$, and an edge cost c_{ij} associated with each edge $(i, j) \in E$. The MINNEWCP problem is to select a subset of the vertices $P \subseteq V$ that minimize the difference between the cost of the edges in $E(P)$ and the weight of the vertices in P. In other words, the objective is to minimize the quantity $\sum_{i,j \in P} c_{ij} - \sum_{v \in P} \pi_v$. In our pricing problem we have an additional cardinality constraint on P, $|P| = S$.

The MINNEWCP is equivalent to the generalized independent set problem first defined by Hochbaum & Pathria (1997). For the equivalence, see Ji & Mitchell (2005). And it gives one way to show that our pricing subproblem, MINNEWCP with cardinality constraints, is an NP-hard problem.

In k-way equipartition, it is possible to change the node-edge-weighted clique problem into an edge-weighted problem. Since every feasible cluster has exactly S vertices, we can shift the node weights to the edge weights in the following way: change the node weights to zero, and replace every edge weight $c(i, j)$ by $c'(i, j) = c(i, j) + \pi_i/(S - 1) + \pi_j/(S - 1)$. In this new graph, the total edge weight of a clique of size S is the same as the total node and edge weight in the original graph. It looks like this conversion would simplify the pricing problem, but because of two reasons, we will stay with our original node-edge-weighted cluster problem model. First, after using our special branching strategy, explained in section 5, this edge-weighted clique problem formulation is not valid any more in non-root nodes. Second, previous research on the maximum edge-weighted clique problem without cardinality constraints shows that an integer programming problem with only binary variable corresponding to the edges performs much worse than the extended formulation with binary variables for both vertices and edges.

G. Dijkhuizen (1993) formulated the maximum edge-weighted clique problem without cardinality constraints as an integer programming problem using only binary variables corresponding to the edges. They investigated the polyhedral structure, and used a cutting-plane approach to solve the problem. The largest instance they solve refer to a graph of 25 nodes and extremely poor performance is reported for quite smaller instances, so the authors concluded that the cutting-plane approach was not suitable to solve the maximum edge-weighted clique problem. In comparison, the computation results on an extended formulation, where binary variables are defined for both the edges and nodes, reports much better results (see Park et al. 1996). In view of these previous work, our integer programming formulation will use binary variables for both edges and vertices, even though at the root node, we can convert the problem to a minimum edge-weighted clique problem.

Another related problem is the pricing problem in Mehrotra & Trick (1998) for their knapsack clustering problem. The only difference is that they did maximization of the same objective function with a knapsack upperbound constraint rather than a fixed constraint on the size. They proposed an effective combinatorial method to solve it, which leads to the success of the main price-and-branch scheme. The strength of their method lies in a shifting of weight from edges to node to give a close upper bound. This way they can get a good upper bound from a combinatorial analysis rather than solving a LP. But we can not make use of this weight shifting scheme to get an useful bound for our minimization problem.

Additional constraints in the pricing problem resulted from our cutting planes and size constraints also impose further difficulties in applying a combinatorial method. In an earlier paper, Johnson et al. (1993) worked on the the same problem as in Mehrotra & Trick (1998): min-cut clustering with capacity lower bound. They looked at the subproblem as a integer programming problem and gave some strong valid inequalities for the subproblem. Again because the difference on the objective, these valid inequalities are usually not violated in our pricing problem.

4.2 Heuristic Methods

Before trying to solve the problem to optimality, we first try some heuristic algorithms to generate good solutions as new columns to price in. Algorithms 1-3 give the three heuristics that we employed. They are applied in the order of 1 to 3. Only when the previous algorithms do not generate any columns to be priced in, would we try the next algorithm. But we need to note here that it is necessary to run algorithm 2 before the solving the problem as a IP problem, so that we can correctly incorporate the cutting planes into the branch-and-price scheme.

Algorithm 1 Heuristic Pricing Algorithm I: Enumerate from 2S closest vertices

```
for $i = 1$ to $n$ do
  Find the closest $2S$ vertices to vertex $i$.
  Enumerate all clusters of size $S$ from these $2S$ vertices.
  Put the violating clusters into a column pool.
end for
Add the 10 most violating columns from the column pool, with no more than 10
columns on the same vertex added.
```

Algorithm 2 Heuristic Pricing Algorithm II: Enumerate from constraint clusters

```
for $i = 1$ to $q$ do
  Enumerate all subsets of $Q_i$ of size $S$.
  Put the violating clusters into a column pool.
end for
Add the 10 most violating columns from the column pool, with no more than 10
columns on the same vertex added.
```

4.3 IP Formulation

When heuristic algorithms for the node-edge-weighted clique problem do not generate improving clusters, we have to find a way to either generate an improving cluster or show that no such clusters exist.

Let c_{ij} in C to be the edge weight for edge (i, j), $\pi_i, i = 1, 2, ...n$, to be the dual variables for (9) in the current RMLP has $\pi_i, i = 1, 2, ...n,$. To incorporate the column generation step into the branch-and-price framework later, we also suppose each vertex v has a cardinality a_v. At the root node, $a_v = 1$.

Define a variable y_i for each vertex $i \in V$, and a variable $z_{ij} = y_i y_j$ for every edge $(i, j) \in E$.

Algorithm 3 Heuristic Pricing Algorithm III: Greedily find a small node-edge-weighted clique of size at least S

```
Input:
 1: $S$ - minimum cluster size
 2: $c_{ij}$ - edge weight
 3: $\pi_i$ - node weight
 4: $a_i$ - node capacity
Steps:
 5: for $i = 1$ to $N$ do
 6:   $CLIQ = \{i\}$
 7:   Find vertex $v$ s.t. $w(CLIQ, v) = \pi_v - \sum_{e \in \delta(v, CLIQ)} c_e$ is minimum for $v \notin CLIQ$.
 8:   if $|CLIQ| < S$ or $w(CLIQ, v) > 0$ then
 9:     $CLIQ = CLIQ + v$
10:     GOTO 7
11:   else
12:     GOTO 14
13:   end if
14:   Check if $CLIQ$ can be put into the violating column pool.
15:   Do local search near $CLIQ$, see if any columns can be put into violating
      column pool.
16: end for
17: Add the 10 most violating columns from the column pool, with no more than
    5 columns on the same vertex added.
```

$$y_i = \begin{cases} 1 & \text{if } i \in P \\ 0 & \text{otherwise} \end{cases} \qquad z_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E(P) \\ 0 & \text{otherwise} \end{cases}$$

We can formulate MINNEWCPQP as the following IP problem, referred to as MINNEWCPIP,

$$\begin{aligned}
\min \quad & -\sum_{i \in V} \pi_i y_i + \sum_{(i,j) \in E} c_{ij} z_{ij} & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i y_i = S & & (14) \\
& \sum_{i \notin Q_k} a_i y_i \ge 1 & k = 1..q \quad & (15) \\
& z_{ij} \le y_i & & (16) \\
& z_{ij} \le y_j & & (17) \\
& z_{ij} \ge y_i + y_j - 1 & & (18) \\
& z_{ij} \ge 0 & & (19) \\
& y_i \in \{0,1\}, z_{ij} \in \{0,1\} & & (20)
\end{aligned}$$

Constraint (14) imposes the size constraint of feasible clusters. Constraint (15) is from the discussion in the last section, to enforce that the selected cluster must not be included in any $Q_i, i = 1..q$.

Equations(16) and (17) ensure that z_{ij} must be zero if either one of y_i or y_j is zero. Equation (18) ensures that z_{ij} is one if both y_i and y_j are one. The convex hull of (16), (17),(18), (19) (20) is called the Boolean Quadratic Polytope (see Padberg 1989). Adding on the an additional size constraint $\sum_{i=1}^{n} y_i = S$, we get the polytope corresponding to our problem. This additional constraint comes from constraint (14) at root node, when $a_i = 1$. a_i may change in the branch-and-bound tree.

Since c_{ij} is always nonnegative, we don't need constraints (16)-(17). Also we don't need to require z_{ij} to be a binary variable explicitly. Because $a_i \geq 1, i = 1, \ldots, n$, we can drop the a_i in constraint (15). So we can reformulate the problem as the following (PRICEIP),

$$\begin{aligned}
\min \quad & -\sum_{i \in V} \pi_i y_i + \sum_{(i,j) \in E} c_{ij} z_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i y_i = S && (21) \\
& \sum_{i \notin Q_k} y_i \geq 1 \qquad k = 1..q && (22) \\
& z_{ij} \geq y_i + y_j - 1 && (23) \\
& z_{ij} \geq 0 \qquad (i,j) \in E && (24) \\
& y_v \in \{0,1\} \qquad v \in V && (25)
\end{aligned}$$

We simply use a branch-and-bound procedure to solve PRICEIP directly. As discussed in the previous section, if the optimal value of this pricing problem is smaller than 0, then we can generate new columns, otherwise, we have achieved optimality in the master problem. It is not necessary to generate the column with the most negative reduced cost each time. Any column with a negative reduced cost would do, so we stop the IP solver as soon as we find a column to price in. But if eligible columns to be priced in do not exist, we have to solve the pricing problem to optimality to prove so.

5 Branching

Branching rules in the column generation context are quite different from those in a branch-and-cut scheme. The simple 0-1 branching rule in a branch-and-cut framework would not work here, Suppose a certain column is fixed at zero at a particular branch, we have to prevent this cluster from being regenerated when generating columns in the branch-and-bound nodes in this branch. This, in general, would lead to finding the k^{th} best subproblem solution rather than the optimal solution, which is a much more expensive operation (see Savelsbergh 1997, Ribeiro et al. 1989). So we need a branching rule that can be easily incorporated in the column generation process.

In the set partition model, the Ryan-Foster branching rule proposed by Ryan & Foster (1981) is commonly used. In a fractional solution for MLP, we can identify two vertices i and j, such that $i, j \in P_1$ and only one of i, j is in P_2, but both x_{P_1} and x_{P_2} are positive. So we can divide the problem into two branches. In one, vertex i and vertex j must be covered by the same cluster, which can be enforced by just collapsing i and j into a single node in the graph. In the other, vertex i and j must be in different clusters, which can be enforced by changing the weight on edge (i, j) to a very large value. This way, we can impose the branching choices on the pricing subproblems directly rather than adding in additional constraints on the master problem.

Notice this branching strategy actually corresponds to the branching strategy on x_{ij} in the compact branch-and-cut formulation of the problem in Mitchell (2001). This conforms with the observation in Lübbecke & Desrosiers (2002), "to branch on meaningful variable sets". Our most valuable source of information are the original edge variables of the compact formulation; they must be integer, so these are what we branch on. Similarly, the cutting planes that we introduced earlier, corresponds to the pigeon constraint for a compact formulation for CPPMIN in Ji (2004, chap. 2).

After branching, our k-way equipartition problem changes into generalized k-way equipartition problem. Each combined vertex v has a vertex weight a_v bigger than one, while each original vertex still has a vertex weight of one. Similarly, in the pricing subproblem, the corresponding a_v are not 1 any more. So we are in fact solving a generalized k-way equipartition problem at every branch-and-bound node except the root node.

Even though we didn't add cutting planes in the sub-nodes in our computational results, the constraints can be easily modified to accommodate this change on the cardinality on the vertices, by changing to the following form:

$$\sum_{i:P_i \subseteq Q} x_i \leq k-1 \tag{26}$$

for Q with $\sum_{j:j\in Q} a_j < kS$.

6 Stabilization

Recall that the objective function of the pricing subproblem depends on the dual variable of the solution of (7). The primal problem (6) is very degenerate, since there are $|V|$ rows, but in a feasible integer solution, only $|V|/S = k$ number of $x's$ would be nonzero. Primal degeneracy is well known to cause slow convergence, see Gilmore & Gomory (1963). Various methods has been proposed on this issue to reduce oscillations that results in useless moves in the dual space, see Lübbecke & Desrosiers (2002).

Instead of using those more complicated methods in the above literature, we simply try to reduce oscillations by starting with more columns than necessary in our initial problem. To guarantee our initial problem is feasible, we include the columns of a good heuristic solution into the initial columns. To help stabilizing the dual variables, we also include the clusters consisting of the close $S-1$ vertices to each vertex. Our choice of using heuristic Algorithm 1 first in searching for violated columns also helps to avoid generating useless columns.

7 Algorithm Outline

In this part, we are going to give an outline for our branch-and-price-and-cut algorithm.

We first use a heuristic algorithm, Algorithm 4, adopted from Domingo-Ferrer & Mateo-Sanz (2002) to find a good feasible solution. An initial problem including the clusters in this solution would be a feasible problem. The heuristic algorithm is coded in FORTRAN. To improve stability of the dual variable value, in the initial formulation, we also include the clusters composed of $S-1$ close neighbors to every vertex .

Algorithm 4 A Heuristic Algorithm for k-way equipartition

```
1: Let $U = V$.
2: Find the two most distant vertices $x_s, x_t \in U$.
3: Form a cluster around $x_s$ with the $S-1$ closet vertices to $x_s$ in $U$, remove these
   $S$ vertices from $U$.
4: Form a cluster around $x_t$ with the $S-1$ closet vertices to $x_t$ in $U$, remove these
   $S$ vertices from $U$.
5: If $|U| \geq 2S$, go back to step 2.
6: Try to improve the solution locally by switching vertices and moving extra
   vertices around.
```

The branch-and-price-and-cut code is implemented using MINTO 3.0.2. The algorithm follows the framework as in MINTO (Nemhauser et al. 1994) with some changes. It is illustrated in Algorithm 5. One major difference is in step 4 and 5. Since step 5, solving the pricing problem as an IP, is time consuming, we try to generate cutting planes before step 5. It is only when we can neither generate new columns using other methods, nor generate cutting planes, would we start solving the pricing problem as an IP.

Algorithm 5 Branch and Price and Cut Framework

```
1: Initialize.
2: Approximately solve the current LP relaxation using CPLEX.
3: Generate columns using heuristic algorithms, if new columns are found goto 2.
4: Check if any cutting planes can be generated, if yes, generate cutting planes
   and goto 2.
5: Generate columns using an IP solver, if new columns are found goto 2.
6: If the gap between the value of the LP relaxation and the value of the incumbent
   integer solution is sufficiently small, STOP with optimality.
7: Try to improve the incumbent solution locally by switching vertices and move
   extra vertices around.
8: Check if any cutting planes can be generated, if yes, generate cutting planes
   and goto 2.
9: Branching.
```

8 Computational Results

Our computational results consist of 2 parts: we first show the realignment results for three north America sports team realignment problems, then we discuss the performance of the algorithm on larger random instances, generated to simulate microaggregation problems. All experiments are done on a Sun Ultra 10 Workstation.

8.1 Realignment Problems

First, we consider the realignment for three major professional sports leagues: NBA, NHL and NFL. In all three sports, we assume that any team can be assigned to any division. However, we would like to mention here that it is not difficult to include additional restrictions to keep some teams in the same division or some teams apart. To make two teams to be in the same divisions, we can change the value of their distance to 0. To divide two teams in separate divisions, we can change their distance to a very large value. These extra constraints generally make the problem easier to solve mathematically. All three problems are solved fairly quickly at the root node, using 11, 20 and 2 seconds respectively for NBA, NHL and NFL.

In Baseball, the Major League Baseball (MLB) in the United States compose of two leagues, the American League of 14 teams, and the National League of 16 teams. Since the two leagues play under different rules, it is difficult to perform a realignment including all the 30 teams in MLB. Within each individual league, it is not feasible to realign them into divisions of equal size either, making it impossible to model it as k-way equipartition problem. Instead, the CPPMIN model can be used to do this realignment to divide the American League into divisions of at least 4 teams, and the National league of divisions of at least 5 teams, see Ji & Mitchell (2005). Currently, the American League is divided into divisions of 5, 5 and 4 teams. The National League is divided into divisions of 5, 5 and 6 teams.

8.2 NBA

In 2004, NBA realigned their $n = 30$ teams into eastern and western conference. Each conference has a total of 15 teams with 3 divisions of $S = 5$ teams. Teams will play divisional opponents four times each, conference opponents outside the division three or four times each and opponents outside the conference two times each, see NBA (2003).

The minimum sum of intradivisional travel distance is 40487km, achieved by the arrangement illustrated in Figure 3 and listed in Table 1. The problem is solved at root node, using 11 seconds. The branch-and-price-and-cut code added 3 constraints, and 79 variables in totally 16 stages, i.e. 15 (RMLP) was solved. The IP solver for the pricing problem was called twice.

The NBA choice is shown in Table 2 with a sum of intradivisional travel distances of 40524km. The NBA choice is different from our optimal solution by 3 cities, Washington DC, Toronto and Indiana. This difference results in a 0.089% increase on the total travel distance.

8.3 NHL

Currently, NHL has $n = 30$ teams divided into eastern and western conferences, each has a 3 divisions of $S = 5$ teams. Eastern Conference clubs play other clubs in its division a total of five times. They play clubs in the other divisions in its conference a total of four times and play either one or two games against all clubs outside its conference. Western Conference clubs face each divisional opponent six times. They play clubs in the other divisions in its conference a total of four times and play either one or two games against all clubs outside its conference.

The minimum sum of intradivisional travel distance is 43146km, achieved by the arrangement illustrated in Figure 4 and listed in Table 3. The problem is solved at root node, using 20 seconds. The branch-and-price-and-cut code added 6 constraints, and 56 variables in the 16 stages. The IP solver for the pricing problem is only called once, to prove the optimality of the MLP solution.

The NHL choice is shown in Table 4 with a sum of intradivisional travel distances of 46443km. The NHL choice is different from our optimal solution by 3 cities, Columbus, Washington DC and Pittsburgh, This difference results in a 7% increase on the total travel distance.

8.4 NFL

In 2002, the NFL realigned its $n = 32$ teams into 8 divisions, each containing $S = 4$ teams. Each team plays sixteen games during the regular season, each team plays each other team in its division twice, playing its remaining games against a subset of the teams outside its division, see NFL (2001). The minimum sum of intradivisional travel distance is 27967km, as found previously in Mitchell (2003). This compares with the NFL choice with a sum of intradivisional travel distances of 50480km. The NFL choice is designed to keep several traditional rivalries that have developed over many years. Our branch-and-price-and-cut code solved the problem at the root node in 2 seconds. It added 12 constraints and 44 variables in 16 stages, running the IP solver for the pricing problem once.

8.5 Micro Aggregation Problems

Another application for k-way equipartition is micro-aggregation problems. Micro-aggregation is a technique to process statistical data for releasing to the public. It is used for economic data where respondent identifiability is high. To protect the confidentiality of each individual respondent, data are divided into groups with minimum size, then the statistics for each group, rather than each individual, is released. It is desirable to group similar respondents together to preserve the informational content of the data as much as possible. For univariate data, the data can be sorted and the problem can be solved efficiently using a dynamic programming approach, see Hansen & Mukherjee (2003), but for higher dimensional data, it becomes a hard problem since sorting is no longer well defined. Domingo-Ferrer & Mateo-Sanz (2002) surveyed the heuristic methods used for micro-aggregation, while Sande (2002) surveyed other methods as well. In certain situations, the groups have equal cardinality. It then can be described as a k-way equipartition problem.

Economic data for micro-aggregation problems tends to be highly skewed with a long tail. So we generate the experiment data in the following way, as suggested by Sande (2003). Vertices are randomly generated points on a 2-dimensional plane. Both x and y coordinates follow an exponential distribution independently. The edge weight is the integral part of the Euclidean distance between vertices. An example of the solution on this kind of data is shown in Figure 2.

Table 5 shows the results for graphs with $S = 4$, graph size n ranging from 40 to 100. The table is divided into three parts to represent the performance of the heuristic algorithm, the root node of the branch-and-price-and-cut algorithm, and the branch-and-price tree. They are labelled as "Heuristic Alg", "Root Node" and "B&P" respectively.

The first block gives the gap and time for the heuristic approach. The second block corresponds to the result at the end of the root node before branching. The first three rows in this block give the total number of instances, and out of these many instances, how many are solved to optimality in the root node, how many get better solutions than the heuristic solution. The rest of the data in this block are all average performance, including, the gap at the end of root node, the running time (in seconds), the number of LP's solved, the number of cuts added, the total number of columns at the end of the root node, the number of columns in the initial formulation, the total number of columns found by solving the pricing problem as an IP, the total time spent on solving the pricing problem as an IP and finally the time for solving one pricing problem as an IP.

For problems not solved to optimality, we start branching. No cutting planes are added any more. Since this step is time consuming, we give an upper bound of 10 on the number of branch-and-bound nodes. The performance is recorded in the third block, including the total number of instances requiring branching, the number of instances solved to optimality before reaching the upper bound of 10 for the total number of nodes (notice this number does not include the problems that are already solved at the root node), how many root node solutions get improved during the branching stage, the final gap between the best IP solution and the LP lower bound, total running time (including the root node time), the number of branch and bound nodes, and the overall number of columns generated in the whole tree.

The tables shows that we are able to solve problems of size $n = 100$ in 2 minutes with a gap of 1.38%. For the problems up to size $n = 100$, our performance is slightly better than the results in Mitchell (2001), which used a branch-and-cut method for uniformly distributed vertices on a square. The small gap at the end of root node indicates a tight relaxation from the combination of column generation and row generation. However, note that the time it takes to solve the root node scales up pretty quickly. For problems smaller than 43 vertices, they are solved to optimality pretty quickly (within 20 seconds). But for problems bigger than 60 vertices, the computation time increases dramatically. This is due to the increasing solution time for the pricing problem, and the increasing number of pricing problems we need to solve. Even though the computation time for solving one pricing problem as an IP is not very big, considering that these pricing problems are NP hard problems, solving them repeatedly sums up to a pretty long time.

9 Conclusions

In this paper, we modelled the sports team realignment problem as k-way equipartition problem and solved the realignment for NBA, NHL and NFL. For

NBA and NHL, we found optimal realignments that improve the current realignment in terms of total intradivisional travel distance. For NFL, assuming any team can be assigned to any division, we confirmed the optimal solution found in Mitchell (2003).

The k-way equipartition problem is solved using branch-and-price scheme. We successfully included cutting planes in the branch-and-price framework. The pricing subproblem is solved as an integer programming problem.

Our computational results on larger instances showed that branch-and-price performed well on small-size instances (within around 40 vertices), but ran into difficulty for larger problems due to the lack of efficient methods for the pricing subproblem. However, the root algorithm gave a good feasible solution most of the time. We conclude that the performance of branch-and-price-and-cut is comparable with the performance of branch-and-cut method in Mitchell (2001).

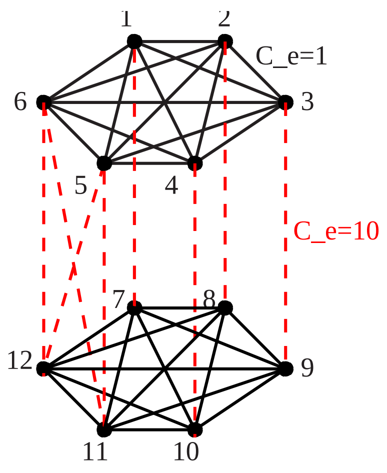

Figure 1 An example on K_{12}

Division	Teams		
1	Boston(2)	New Jersey(18)	New York(20)
	Philadelphia(22)	**Washington DC(30)**	
2	Chicago(4)	Cleveland(5)	Detroit(8)
	Milwaukee(16)	**Toronto(28)**	
3	Atlanta(1)	Charlotte(3)	Miami(15)
	Orlando(21)	**Indiana(11)**	
4	Denver(7)	Minnesota(17)	Portland(24)
	Seattle(27)	Utah (29)	
5	Golden State(9)	LA Clippers(12)	LA Lakers(13)
	Phoenix(23)	Sacramento(25)	
6	Dallas(6)	Houston(10)	Memphis(14)
	New Orleans(19)	San Antonio(26)	

Table 1 Optimal Alignment for NBA, total intradivisional distance = 40488km

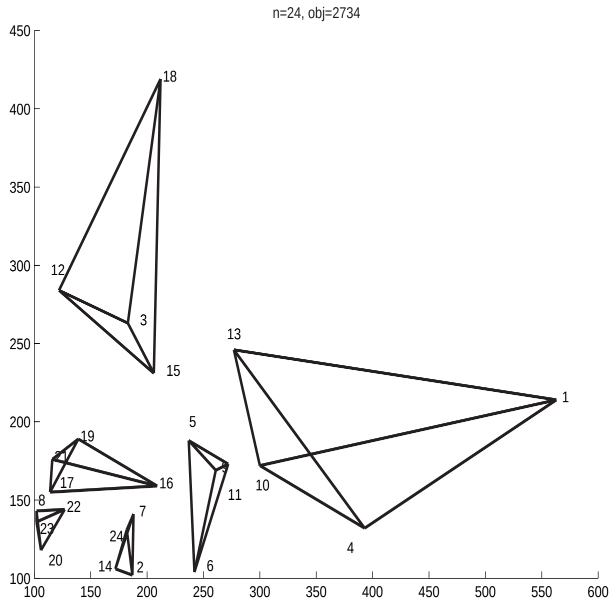

Figure 2 An example of micro-aggregation data: $n = 24, S = 4$

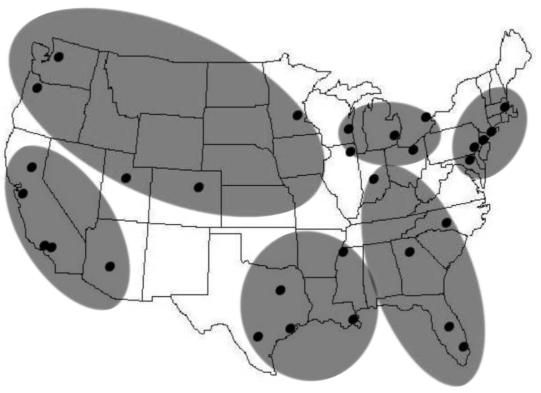

Figure 3 The realignment that minimizes the sum of intradivisional travel distances for NBA

References and Notes

1 Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P. & Vance, P. H. (1998), 'Branch-and-price: column generation for solving huge integer programs', *Operations Research* **46**, 316–329.

2 Domingo-Ferrer, J. & Mateo-Sanz, J. M. (2002), 'Practical data-oriented microaggregation for statistical disclosure control', *IEEE Transactions on Knowledge and Data Engineering* **14**(1), 189–201.

3 G. Dijkhuizen, U. (1993), 'A cutting-plane approach to the edge-weighted maximal clique problem', *European Journal of Operational Research* **69**, 121–130.

Division	Teams		
Atlantic	Boston(2)	New Jersey(18)	New York(20)
	Philadelphia(22)	**Toronto(28)**	
Central	Chicago(4)	Cleveland(5)	Detroit(8)
	Milwaukee(16)	**Indiana(11)**	
Southeast	Atlanta(1)	Charlotte(3)	Miami(15)
	Orlando(21)	**Washington DC(30)**	
Northwest	Denver(7)	Minnesota(17)	Portland(24)
	Seattle(27)	Utah (29)	
Pacific	Golden State(9)	L.A. Clippers(12)	L.A. Lakers(13)
	Phoenix(23)	Sacramento(25)	
Southwest	Dallas(6)	Houston(10)	Memphis(14)
	New Orleans(19)	San Antonio(26)	

Table 2 Alignment chosen by the NBA, total intradivisional distance = 40524km

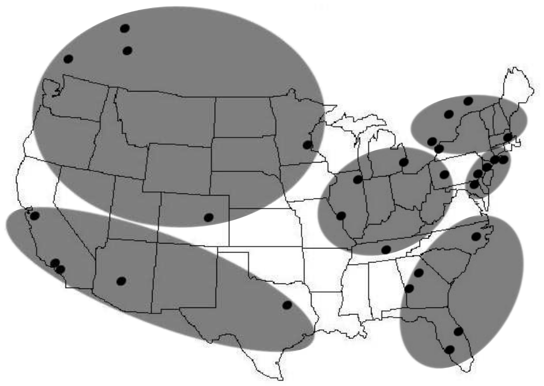

Figure 4 The realignment that minimizes the sum of intradivisional travel distances for NHL

4 Gilmore, P. & Gomory, R. (1963), 'A linear programming approach to the cutting stock problem – part *ii*', *Oper. Res.* **11**, 863–888.

5 Guttmann-Beck, N. & Rubinstein, S. (1998), 'Approximation algorithms for minimum tree partition', *Discrete Applied Mathematics* **87**(1-3), 117–137.

6 Hansen, S. L. & Mukherjee, S. (2003), 'A polynomial algorithm for optimal microaggregation', *IEEE transactions on Knowledge and Data Engineering* **15**(1), 1043–1044.

7 Hochbaum, D. S. & Pathria, A. (1997), 'Forest harvesting and minimum cuts: A new approach to handling spatial constraints', *Forest Science* **43**(4), 544–554.

8 Ji, X. (2004), Graph partition problems with minimum size constraints, PhD thesis, Rensselaer Polytechnic Institute.

9 Ji, X. & Mitchell, J. E. (2005), Graph partition problems with minimum size constraints using branch-and-price, Technical report, Rensselaer Polytechnic Institute. forthcoming.

10 Johnson, E. L., Mehrotra, A. & Nemhauser, G. L. (1993), 'Min-cut clustering', *Mathematical Programming* **62**, 133–151.

Division	Teams		
1	Atlanta(2)	Carolina(6)	Florida(13)
	Tampa Bay(27)	**Columbus(9)**	
2	NY Rangers(17)	NY Islanders(18)	New Jersey(20)
	Philadelphia(22)	**Washington DC(30)**	
3	Boston(3)	Buffalo(4)	Montreal(16)
	Ottawa(21)	Toronto(28)	
4	Chicago(7)	Detroit(11)	Nashville (19)
	St. Louis(26)	**Pittsburgh(24)**	
5	Calgary(5)	Colorado(8)	Edmonton(12)
	Minnesota(15)	Vancouver(29)	
6	Anaheim(1)	Dallas (10)	Los Angeles(14)
	Phoenix(23)	San Jose(25)	

Table 3 Optimal Alignment for NHL: total intradivisional distance = 43146km

Division	Teams		
Southeast	Atlanta(2)	Carolina(6)	Florida(13)
	Tampa Bay(27)	**Washington DC(30)**	
Atlantic	NY Rangers(17)	NY Islanders(18)	New Jersey(20)
	Philadelphia(22)	**Pittsburgh(24)**	
Northeast	Boston(3)	Buffalo(4)	Montreal(16)
	Ottawa(21)	Toronto(28)	
Central	Chicago(7)	Detroit(11)	Nashville (19)
	St. Louis(26)	**Columbus(9)**	
Northwest	Calgary(5)	Colorado(8)	Edmonton(12)
	Minnesota(15)	Vancouver(29)	
Pacific	Anaheim(1)	Dallas (10)	Los Angeles(14)
	Phoenix(23)	San Jose(25)	

Table 4 Alignment chosen by the NHL: total intradivisional distance = 46443km

11 Lisser, A. & Rendl, F. (2003), 'Graph partitioning using linear and semidefinite programming', *Mathematical Programming* **95**(1), 91–101.

12 Lübbecke, M. E. & Desrosiers, J. (2002), Selected topics in column generation, Technical Report 008-2004, Braunschweig University of Technology.

13 Mehrotra, A., Johnson, E. L. & Nemhauser, G. L. (1998), 'An optimization based heuristic for political districting', *Management Science* **44**(8), 1100–1114.

14 Mehrotra, A. & Trick, M. A. (1996), 'A column generation approach for graph coloring', *INFORMS J. Comput.* **8**, 344–354.

15 Mehrotra, A. & Trick, M. A. (1998), 'Cliques and clustering: a combinatorial approach', *Operations Research Letters* **22**, 1–12.

16 Mitchell, J. E. (2001), Branch-and-cut for the k-way equipartition problem, Technical report, Rensselaer Polytechnic Institute.

17 Mitchell, J. E. (2003), 'Realignment in national football league: Did they do it right', *Naval Research Logistics* **50**(7), 683–701.

18 NBA (2003), 'NBA approves realignment for 2004-05 season'. The National Basketball

n	40	60	80	100
$k = \lfloor \frac{n}{S} \rfloor$	10	15	20	25
Heuristic Alg.				
Gap	6.05%	3.27%	4.66%	5.85%
Time	0.0106	0.0199	0.0329	0.0458
Root Node				
Total Instances	5	5	5	5
Solved exactly	5	1	3	1
Better Solution	5	4	4	5
Gap	0.00%	0.31%	0.25%	1.38%
Time	19.79	150.33	103.18	118.50
LPs solved	14	57	30	51
Total Cuts	7	45	23	50
Total Columns	180	315	356	466
Initial Columns	127	179	247	319
by IP	1	5	2	0
Total IP Time	19.40	116.00	98.60	118.50
Avg IP Time	9.70	20.17	32.87	69.50
B &P run				
total instances	0	4	2	4
Solved exactly	0	3	1	3
Better Solution	0	1	1	2
Gap	0.00%	0.09%	0.24%	0.31%
time	19.79	233.81	446.60	1182.62
nodes	1	4	3	8
Final columns	180	325	364	508

Table 5 Branch-and-Price Results on k-way Equipartition for Microaggregation Problems for $S = 4$

Association.
URL: *http://www.nba.com/news/realign_031117.html*

19 Nemhauser, G. L. & Park, S. (1991), 'A polyhedral approach to edge coloring', *Operations Research Letters* **10**, 315–322.

20 Nemhauser, G. L., Savelsbergh, M. W. P. & Sigismondi, G. C. (1994), 'MINTO, a mixed INTeger optimizer', *Operations Research Letters* **15**, 47–58.

21 NFL (2001), 'New century, new look for NFL'. The National Football League.
URL: *http://www.nfl.com/news/010522realignment.html*

22 Padberg, M. (1989), 'The boolean quadric polytope: Some characteristics, facets and relatives', *Mathematical Programming* **45**, 139–172.

23 Park, K., Lee, K. & Park, S. (1996), 'An extended formulation approach to the edge-weighted maximal clique problem', *European Journal of Operational Research* **95**, 671–682.

24 Ribeiro, C., Minoux, M. & Penna, M. (1989), 'An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment', *European J. Oper. Res.* **41**, 232–239.

25 Ryan, D. M. & Foster, B. A. (1981), An integer programming approach to scheduling, *in* A. Wren, ed., 'Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling', Amsterdam, North-Holland, pp. 269–280.

26 Sande, G. (2002), ‘Exact and approximate methods for data directed microaggregation in one or more dimensions’, *International Journal of Uncertainty,Fuzziness and Knowledge-Base Systems* **10**(5).

27 Sande, G. (2003), ‘Personal communication’.

28 Savelsbergh, M. (1997), ‘A branch-and-price algorithm for the generalized assignment problem’, *Operations Research* **45**, 831–841.

29 Vance, P. H., Barnhart, C., Johnson, E. L. & Nemhauser, G. L. (1997), ‘Airline crew scheduling: A new formulation and decomposition algorithm’, *Operations Research* **45**, 188–200.

30 Vanderbeck, F. (1999), ‘Computational study of a column generation algorithm for bin packing and cutting stock problems’, *Mathematical Programming* **86**(3), 565–594.

31 Wilhelm, W. E. (2001), ‘A technical review of column generation in integer programming’, *Optimization and Engineering* **2**, 159–200.