WORKING PAPER L-2005-02

WP

Michael M. Sørensen

Polyhedral Computations for the Simple Graph Partitioning Problem

Logistics/SCM Research Group

Department of Accounting, Finance and Logistics

Polyhedral Computations for the Simple Graph Partitioning Problem

Michael M. Sørensen
Aarhus School of Business
Dept. of Accounting, Finance and Logistics
Fuglesangs Allé 4
DK-8210 Aarhus V
Denmark
email: mim@asb.dk

November 3, 2005

Abstract

The simple graph partitioning problem is to partition an edge-weighted graph into mutually disjoint subgraphs, each containing no more than b nodes, such that the sum of the weights of all edges in the subgraphs is maximal. In this paper we present a branch-and-cut algorithm for the problem that uses several classes of facet-defining inequalities as cutting-planes. These are b-tree, clique, cycle with ear, multistar, and S,T-inequalities. Descriptions of the separation procedures that are used for these inequality classes are also given. In order to evaluate the usefulness of the inequalities and the overall performance of the branch-and-cut algorithm several computational experiments are conducted. We present some of the results of these experiments.

Key words: Branch-and-cut algorithm, Facets, Graph partitioning, Multicuts, Separation procedures.

1 Introduction

Given an edge-weighted graph G on n nodes and a positive integer $b \leq n$, the simple graph partitioning problem (SGPP) is to determine a partition of G into subgraphs, each containing at most b nodes, such that the sum of the weights of all edges in the subgraphs is maximal. The set of nodes in a subgraph of a partition is called a *cluster*. This problem is $\mathcal{NP}$-hard in general when $b \geq 3$; it specializes to a weighted matching problem when $b = 2$. In this paper we consider the computational aspects of a polyhedral approach to the SGPP. In particular, we evaluate the usefulness of several classes of inequalities that define facets of the associated simple graph partitioning polytope, and we provide some computational results of a branch-and-cut algorithm for the SGPP.

We use the following notation in this paper. The complete graph on n nodes is denoted by $K_n = (V_n, E_n)$. Let $S, T \subset V_n$ be two disjoint subsets of nodes. $\delta(S,T)$ is the set of edges in E_n with one endnode in S and the other endnode in T; however, instead of $\delta(\{s\}, V_n \setminus \{s\})$ we use $\delta(s)$ to denote the star of node s in E_n. $E_n(S)$ is the set of edges in E_n with both endnodes in S. Let $x \in \mathbb{R}^{E_n}$, let $E \subseteq E_n$, and let $a \in \mathbb{R}$. We write $ax(E)$ as shorthand for the sum $\sum_{e \in E} ax_e$.

We associate the SGPP with the complete graph K_n by giving all edges in E_n that do not exist in G a zero weight so that these edges give no contribution to a partition of K_n. We define variables x_e for all $e \in E_n$ such that $x_{uv} = 1$ if and only if nodes u and v are in the same cluster of a partition; otherwise $x_{uv} = 0$. Given edge weights $c_e \in \mathbb{Z}$ for all $e \in E_n$ and integer cluster capacity $b \geq 3$, the following 0–1 ILP formulation of the SGPP was introduced in [3].

$$
\begin{array}{lll}
\max & \displaystyle\sum_{e \in E_n} c_e x_e & \\
\text{s.t.} & & \\
& x_{uv} + x_{uw} - x_{vw} \leq 1 & \text{for all } (u,v,w) \subset V_n \\
& x(\delta(v)) \leq b - 1 & \text{for all } v \in V_n \\
& x_e \geq 0, \text{ integer} & \text{for all } e \in E_n.
\end{array}
\tag{1}
$$

The first set of constraints, called *triangle inequalities,* ensures that $x \in \{0,1\}^{E_n}$ is an incidence vector of a partition of K_n. There are three distinct triangle inequalities for each triple of nodes. The second set of constraints, the *star inequalities,* imposes the capacity restriction of the clusters. The upper bounds $x_e \leq 1$ on the variables are not necessary in (1) because they are implied by the triangle inequalities.

Since we choose to work with the complete graph, our formulation of the SGPP may have more variables and constraints than what would be necessary if an alternative formulation was used. On the other hand, this means that we can formulate the problem by using only variables that represent the edges. In this way we avoid using any variables that represent the allocation of the nodes to clusters (so-called cluster variables), and thus we avoid introducing symmetry into the formulation; e.g. see [4] and [8].

All inequalities in (1) define facets of the simple graph partitioning polytope under mild conditions. The simple graph partitioning polytope, which we denote by $P_n(b)$, is the convex hull of all vectors that are feasible solutions to (1). The importance of facet-defining inequalities is that they define the boundaries of the polytope, so that they are the tightest possible valid inequalities that can be used to solve the problem. In order to get an even tighter representation

of $P_n(b)$ than that given by the inequalities in (1) we use further facet-defining inequalities that belong to several different classes. This means that a very large number of inequalities is usually employed in the solution process. To get this working in a practical manner we have designed a polyhedral cutting-plane algorithm which is embedded in a branch-and-bound enumeration, thus providing two of the main components of a branch-and-cut algorithm for the SGPP.

Several closely related studies are described in the literature. Faigle et al. [3] were the first to take a polyhedral approach to the SGPP. Barahona et al. [1] use a cut-and-branch algorithm to solve the max-cut problem, Brunetta et al. [2] solve the equipartition problem by branch-and-cut, Ferreira et al. [4, 5] use branch-and-cut to solve a node and edge-weighted version of the graph partitioning problem, and Grötschel and Wakabayashi [6, 7] use a cutting-plane algorithm for the clique partitioning problem. A different approach to solving graph partitioning problems is described by Johnson et al. [9] who use a column generation algorithm.

The rest of this paper is organized as follows. In the next section we describe the separation procedures for generating inequalities of the various classes that are used in the cutting-plane algorithm. Section 3 gives an overview of the branch-and-cut algorithm for the SGPP. Subsequently, in section 4, we consider the computational experiments and present some results. Finally, we make some concluding remarks.

2 Separation procedures

This section describes the separation procedures for the inequalities that are used here as cutting-planes. Given an LP solution $\bar{x}$, the purpose of a separation procedure for a particular class of inequalities is to generate one or more inequalities from the class that are not satisfied by $\bar{x}$. When the separation procedure is guaranteed to identify a most violated inequality, if any exists, it is called an exact separation procedure; when it may fail to find a violated inequality it is a heuristic separation procedure.

The inequalities we use besides those in the ILP formulation belong to the following classes of facet-defining inequalities: S,T-inequalities, cycle with ear inequalities, b-tree inequalities, clique inequalities, and multistar inequalities. We expect that the separation problem of detecting a most violated inequality is $\mathcal{NP}$-hard for all of these inequality classes. For this reason our separation procedures are heuristics that attempt to identify several violated inequalities from each class. The only exceptions are the star and triangle inequalities in (1) which are separated by complete enumeration. Only when these inequalities are satisfied by the current LP solution do we check the other inequality classes. By checking an inequality class we mean that we apply the associated separation procedures in order to generate a number of violated inequalities.

Some of the separation procedures that are described below work on the set of active inequalities in the LP. These are the inequalities which are satisfied at equality by the current LP solution.

2.1 Separation of S, T-inequalities

The following class of S, T-inequalities was introduced in [7] for the clique partitioning polytope $P_n(n)$. Let S and T be two disjoint nonempty subsets of nodes such that $|S| \leq |T|$. Then the S, T-*inequality*

$$x(\delta(S,T)) - x(E_n(S)) - x(E_n(T)) \leq |S|$$

is valid for $P_n(b)$. In particular, when $|S| = 1$ and $|T| = 2$ the inequality specializes to a triangle inequality. The S, T-inequality is facet-defining for $P_n(b)$ when $|S| < |T|$ and $b \geq 4$. In the special case where $b = 3$ the modified S, T-inequality $x(\delta(S,T)) - x(E_n(T)) \leq |S|$ is facet-defining when $3 \leq |S| \leq |T|$ or $|S| < |T|$ when $|S| \leq 2$.

These inequalities are similar in structure to the α-inequalities for the b-clique polytope [10]. Here we use a separation heuristic which is inspired by the one described in [13] for the α-inequalities. The basic idea is to modify currently active S, T-inequalities. Since these inequalities are satisfied at equality it is often easy to detect a modification that yields a new violated inequality. The modification we consider here is to augment either node set S or node set T by one node.

For each active S, T-inequality in the current LP we do the following. For all nodes $v \in V_n \setminus (S \cup T)$ we calculate $\delta_S^v := \bar{x}(\delta(\{v\}, S))$ and $\delta_T^v := \bar{x}(\delta(\{v\}, T))$. If $\delta_S^v > \delta_T^v$ a new violated S, T-inequality is obtained by adding node v to node set T. On the other hand, if $\delta_T^v > \delta_S^v + 1$ a new violated inequality can be obtained by adding v to S. Among the new violated inequalities that are determined in this way we only generate the one that gives the largest violation. The reason for selecting only one violated inequality for each active S, T-inequality is to keep the number of new inequalities that are generated at a manageable level.

In the latter case above where S is augmented it may happen that $|S| = |T|$. In this case the corresponding S, T-inequality is implied by the inequalities that are obtained by removing a node from $S \cup T$ (see the proof of Theorem 4.1 in [7]) and possibly interchanging the roles played by the two node sets. For this reason we look instead for a most violated inequality among the inequalities that are induced by a reduced node set.

2.2 Separation of cycle with ear inequalities

The class of *cycle with ear (CWE) inequalities* was discovered by de Souza and Laurent [15] and Ferreira et al. [4]. Let $G = (V, E)$ be a subgraph on $b+1$ nodes of K_n that has a nondegenerate ear decomposition. That is, $E = C \cup P_1 \cup \cdots \cup P_r$ where C is a cycle and P_i is a path of length at least two whose endnodes belong to $C \cup P_1 \cup \cdots \cup P_{i-1}$ but whose inner nodes do not. P_i is called an ear and its endnodes are denoted by u_i and v_i. The ear decomposition is said to be nondegenerate if $u_i \neq v_i$ for $i = 1, \ldots, r$. Let $A = \{e \in E_n \mid e = u_i v_i \text{ for some } i\}$ be the set of chords associated with the endnodes of the ears, and let g_e be the number of occurrences of e in the list $u_1 v_1, \ldots, u_r v_r$. In [12] it is shown that the cycle with ear inequality for $P_n(b)$ is

$$x(E) - \sum_{e \in A} g_e x_e \leq b - 1.$$

This inequality defines a facet of $P_n(b)$ when $b \geq 3$.

Ferreira et al. [5] use these inequalities and some related inequality classes in their computational study of the node capacitated graph partitioning problem. In particular, they describe two separation heuristics for the CWE inequalities. Here we provide a third separation heuristic. Besides the fact that we work with the complete graph K_n, our heuristic is based on a conceptually different view of the structure of the inequalities. The alternative description of the structure is as follows.

Let $\{w_1, w_2, w_3\} \subset V_n$ be a subset of three distinct nodes. The CWE inequality is obtained as the sum of the inequality $x_{w_1w_2} + x_{w_1w_3} + x_{w_2w_3} \leq 1$ (which is not valid for $P_n(b)$) and $b-2$ triangle inequalities $x_{u_kw_k} + x_{v_kw_k} - x_{u_kv_k} \leq 1$, for $k = 4, \ldots, b+1$, where $u_k, v_k \in \{w_1, \ldots, w_{k-1}\}$ and $w_k \in V_n \setminus \{w_1, \ldots, w_{k-1}\}$. It is easy to see that this is true. The first three nodes constitute an initial cycle, and the support of each triangle inequality either (i) augments the cycle, in which case u_kv_k is an edge of the cycle, (ii) adds a new ear, in which case u_kv_k is not an edge of the cycle or an existing ear, or (iii) augments an existing ear.

This observation immediately suggests a greedy separation heuristic. We enumerate all triangles such that $\bar{x}_{w_1w_2} + \bar{x}_{w_1w_3} + \bar{x}_{w_2w_3} > 1$. To each such triangle we iteratively add $b-2$ further nodes, one node w_k at a time, such that the contribution from $\bar{x}_{u_kw_k} + \bar{x}_{v_kw_k} - \bar{x}_{u_kv_k}$ is maximal, where $u_k, v_k \in \{w_1, \ldots, w_{k-1}\}$ are distinct nodes. If the left-hand side contribution after adding node w_k becomes less than or equal to $k-2$, we skip the current CWE configuration because it will not result in a violated CWE inequality; this is because we are ensuring that all triangle inequalities are satisfied by $\bar{x}$. All violated CWE inequalities that are found in this way are generated.

We use one further separation heuristic for the CWE inequalities. This heuristic works on the currently active CWE inequalities. Each node in the CWE configuration is interchanged with a node not in the configuration that leads to the largest increase of the left-hand side of the CWE inequality. In the special case where the CWE configuration is a proper cycle we also detect attractive 2-exchanges of edges in the cycle in an iterative manner.

2.3 Separation of b-tree inequalities

The *b-tree inequalities* were introduced in [14] for the SGPP, and because the description of these inequalities is quite involved, the reader is referred to [14] for details about the structure of the inequalities. Here we just mention that the inequalities are induced by a configuration which is built upon trees of b edges. This configuration also consists of $b-1$ disjoint sets of additional nodes that play the role of superimposing S,T-inequalities on the tree. The b-tree inequalities define facets of $P_n(b)$, for $b \geq 3$, if and only if the tree is not a star and all sets of additional nodes are nonempty.

The structure of the b-tree inequalities has inspired the derivation of the class of cut-tree inequalities for the b-clique polytope in [13]. Here we use separation procedures for the b-tree inequalities that are similar to the procedures that are described in [13], but with a few additional features. Two procedures are greedy construction heuristics, while a third procedure attempts to modify active b-tree inequalities. The procedures only differ in the determination of a tree, while they use the same method for determining the sets of additional nodes.

Both procedures for building b-trees start with a tree that is a single edge

and iteratively augment the tree by adding one edge at a time until the tree has b edges. In the first procedure only "double star" trees are considered; that is, the tree has exactly two inner nodes, the endnodes of the starting edge. In the other procedure no restrictions are imposed on the structure of the resulting tree. New edges are added to the tree in such a way that the most attractive edge that has one endnode in the tree is chosen next. If the resulting tree turns out to be a star it is not considered further. We apply these procedures to all edges where the associated variable has a positive value. In this way several trees are built, but duplicate trees are eliminated. The modification procedure uses the tree in the support of an active b-tree inequality.

When a tree is available the sets of additional nodes must be determined. This is done in almost the same way as described in [13], so we will not go into details here. There is one exception to that method, however, which we would like to mention. Because the inequality is only facet-defining if all node sets are nonempty, we should make sure that at least one node is included in each of the additional node sets. Therefore we start by assigning one node to each of the node sets. This is done by solving a (unbalanced) transportation problem where each node set demands a node. Subsequently, further nodes may be added to the sets whenever it is attractive to do so. All violated b-tree inequalities found are generated.

2.4 Separation of clique inequalities

Let $S \subseteq V_n$ be a subset of nodes such that $|S| \geq b+2$. In [12] it is shown that the *clique inequality*

$$x(E_n(S)) \leq \left\lfloor \frac{|S|}{b} \right\rfloor \binom{b}{2} + \binom{|S| \bmod b}{2}$$

defines a facet of $P_n(b)$, for $b \geq 3$, if and only if $|S|$ mod b $\neq 0$. We use three separation heuristics for the clique inequalities.

The first heuristic starts with the full clique where $S = V_n$. Subsequently we iteratively remove one node at a time from S until there are only $b+2$ nodes left in the clique. During each iteration the node to be removed from the current S is the node $s \in S$ that has the smallest value of $\bar{x}(\delta(\{s\}, S \setminus \{s\}))$, and we let $S := S \setminus \{s\}$. Whenever S mod $b \neq 0$ and S induces a violated clique inequality the inequality is generated.

The inequalites that are generated by this heuristic tend to be associated with nodes sets S of large cardinalities. For this reason the second heuristic has been designed to detect violated inequalities that are induced by relatively small cliques.

In the second heuristic we do as follows for each node $u \in V_n$. An initial node set S is established which consists of node u and $b-1$ other nodes such that the values $\bar{x}_{us}$ are maximal for $s \in S \setminus \{u\}$. Then S is augmented iteratively, one node at a time, by including the node $v \in V_n \setminus S$ for which $\bar{x}(\delta(\{v\}, S))$ is maximal. Whenever the augmented node set $S := S \cup \{v\}$ contains at least $b+2$ nodes and $|S|$ mod $b \neq 0$ we generate the induced clique inequality if it is violated.

The third heuristic is a modification procedure in which three attempts are made to determine new violated clique inequalities from each active clique inequality: (i) if $|S|$ mod $b \leq b-2$ a node outside the clique is added to node

set S, (ii) if $|S| \bmod b \geq 2$ and $|S| \geq b + 3$ a node is removed from node set S, and (iii) a node in S is interchanged with a node not in S. In each of these three cases only the most violated inequality is generated.

2.5 Separation of multistar inequalities

Let $S, T \subset V_n$ be two disjoint sets of nodes such that $|S| \geq 2$ and $|T| \geq 2$. The subgraph of K_n with node set $S \cup T$ and edge set $\delta(S,T) \cup E_n(S)$ is called a multistar with nucleus S and satellites T. If all edges in $E_n(S)$ are deleted from the multistar we obtain a bipartite subgraph. Here we assume that $|S| \geq 2$ and $|T| = (b-1)|S| + d - 2$, where $d \in \{0,1\}$, and we consider the *multistar inequality*

$$x(\delta(S,T)) + dx(E_n(S)) \leq (b-1)|S| + d - 2.$$

When $d = 0$ it is also called a *bipartite subgraph (BSG) inequality*. In [12] it is shown that the inequaltiy defines a facet of $P_n(b)$ when $b \geq 3$.

For any given nucleus S the best set of satellites is determined exactly by including in T the required number of nodes with largest values $\bar{x}(\delta(\{w\}, S))$, for $w \in V_n \setminus S$. Our heuristic for separating multistar inequalities only considers multistars where the nucleus consists of two or three nodes. Multistars with larger nuclei may be generated by a modification procedure that works on the active multistar inequalities.

The separation heuristic considers all node pairs u, v as a nucleus, provided that $\bar{x}_{uv}$ is fractional. We skip the cases where $\bar{x}_{uv}$ is 0 or 1, because then all multistar inequalities with $S = \{u, v\}$ are satisfied by $\bar{x}$ due to the triangle and star inequalities in (1). We consider both the BSG inequality (the case with $d = 0$) and the multistar inequality ($d = 1$) and generate any violated inequalities.

If none of the inequalities with $S = \{u, v\}$ are violated by $\bar{x}$, but if any one of them is nearly violated, the separation heuristic continues to add a node to the nucleus. That is, provided that multistar inequalities with $|S| = 3$ exist given the values of n and b. Each of the remaining $n - 2$ nodes is considered as a third node in S, and the satellite contributions from all other nodes are calculated. The most attractive of these nodes are included in T for both cases $d = 0$ and $d = 1$. All violated inequalities found in this way are generated.

We also use a modification procedure that works on the active multistar inequalities. The first modification that is considered is to determine the best set of satellite nodes T given the nucleus S in the support of the current inequality. The second and last modification attempts to augment the nucleus S by one node as described above. It is common for all modifications that both types of multistar inequalities are considered, and all violated inequalities found are generated.

3 A branch-and-cut algorithm

In this section we describe the most important components of the branch-and-cut algorithm we have implemented for the SGPP. Basically, a branch-and-cut algorithm is an LP-based branch-and-bound procedure which incorporates a

cutting-plane algorithm for the solution of the LP subproblems. We first outline the cutting-plane algorithm, and since the way we perform branching is nonstandard, we also explain the main features of the branch-and-cut enumeration.

Besides these components our algorithm also uses a heuristic for the SGPP in order to quickly obtain an incumbent feasible partition. The value of this partition is used to prune branches of the branch-and-cut tree until a better partition is possibly found during the enumeration. The reader is referred to [11] for a description of this heuristic.

3.1 The cutting-plane algorithm

The cutting-plane algorithm controls the generation of violated inequalities by the separation procedures. It also manages the addition of these inequalities to the LP, the removal of nonbinding inequalities from the LP, and the storage of nonbinding inequalities for later use.

The separation procedures are often able to detect a large number of violated inequalities. In order to keep this number of inequalities at a managable level we generate at most $40n$ inequalities in total every time the separation procedures are called. From the set of generated inequalities we iteratively add up to 400 most violated inequalities to the LP at a time, then reoptimize the revised LP before adding the next subset of violated inequalities. This continues until all generated inequalities are satisfied by the LP solution. We refer to this iterative process as the *reoptimization loop*.

Because of the large number of inequalities that are involved in the optimization process, it becomes necessary to remove nonbinding inequalities from the LP to keep the basis at a reasonable size. So every time the LP has been reoptimized all inequalities with positive slack are removed from the LP. Some of the inequalities that are removed from the LP may be needed later to reconstruct a subproblem in the branch-and-cut enumeration tree and therefore cannot be completely discarded. These inequalities are stored in a cut-pool from which they can easily be retrieved whenever they are needed. The cut-pool is also used to store inequalities that have been active recently and which are separated by a heuristic procedure — in this way we get easy access to some inequalities that might not be found otherwise.

Some of the separation procedures assume that all inequalities in (1) are satisfied by the current LP solution. In order to ensure this the inequalities are generated in a hierarchical manner. There are two levels in this hierarchy. At the first level we only generate star and triangle inequalities. This is done until an LP solution is obtained which satisfies all these inequalities. If the resulting LP solution is integer the cutting-plane algorithm terminates with the incidence vector of a partition which is optimal for the current subproblem. Otherwise we go to the second level and check the other inequality classes.

At the second level we first add all violated inequalities in the cut-pool to the set of generated inequalities. Next we apply the appropriate modification procedures to the currently active inequalities, and finally the separation procedures for CWE, b-tree, clique, and multistar inequalities are employed. After completion of the reoptimization loop we return to the first level of constraint generation.

The cutting-plane algorithm terminates when no violated inequalities are generated or when tailing-off prevails. In the latter case where successive LP values converge very slowly it is usually better to perform branching than to continue the generation of new inequalities. In the present implementation we detect tailing-off at the beginning of the second level in the cutting-plane algorithm: If the LP value decreases by less than 0.1% during three consecutive entries at the second level, cut generation is terminated.

3.2 Branching

The standard way to perform branching for a 0–1 ILP is to select a variable x_e with a fractional value and create two subproblems, one with the additional constraint $x_e = 0$ and the other with the additional constraint $x_e = 1$. This is a reasonable way to do branching, and it is the way it is done in [2] and [5]. We have chosen a more general way to perform branching for the SGPP which exploits that we optimize on the complete graph K_n; we branch on cliques instead of edges.

Let $S \subseteq V_n$ such that $2 \leq |S| \leq b$. Then the branching decision is to force either all nodes or at most $|S| - 1$ of the nodes in S to be in the same cluster of a partition. This gives the branching constraints $x(E_n(S)) = \binom{|S|}{2}$ and $x(E_n(S)) \leq \binom{|S|-1}{2}$, respectively. Note that standard branching takes place when $|S| = 2$.

We determine the set of clique nodes S to branch on in the following way. Initially we select, among the variables with values closest to 0.5, a single variable x_{uv} which has the objective coefficient with largest absolute value. The initial $S = \{u, v\}$ is then augmented by further nodes in order to create several candidate cliques. This is done in a greedy manner. For each candidate clique we examine the effects on the current LP value of the respective branching constraints by adding each constraint and reoptimizing the resulting LP. Let $z_{\mathrm{U}}(S)$ be the objective value that results from adding the up-branch constraint $x(E_n(S)) = \binom{|S|}{2}$, and let $z_{\mathrm{D}}(S)$ be the value that results from the down-branch constraint $x(E_n(S)) \leq \binom{|S|-1}{2}$. We choose the clique S^* to branch on which gives the smallest upper bounds on the values of the subproblems. That is, any other candidate clique S' has $\lfloor z_{\mathrm{U}}(S') \rfloor \geq \lfloor z_{\mathrm{U}}(S^*) \rfloor$ or $\lfloor z_{\mathrm{D}}(S') \rfloor \geq \lfloor z_{\mathrm{D}}(S^*) \rfloor$. Ties are broken by choosing the clique that has the smallest value of $\max\{z_{\mathrm{U}}(S), z_{\mathrm{D}}(S)\}$.

During the branch-and-cut enumeration the next subproblem to be explored is the one with the largest upper bound on the value of an optimal partition. This best-bound search strategy has the advantage of minimizing the number of subproblems to be enumerated. Among the disadvantages of this strategy is that the LPs in two successive subproblems tend to be highly unrelated, thus requiring relatively long reoptimization times. In order to overcome this weakness we store a list of all binding constraints and basic variables at the parent node in the enumeration tree. So the LP of the parent problem is reconstructed by getting all necessary constraints from the cut-pool, and the associated optimal basic solution is calculated before solving the subproblems.

We also apply variable-fixing as part of the initialization of each subproblem. Here we use logical implications that follow from the triangle inequalities. Suppose that a variable x_{uv} is fixed to a value of 1. If the variable x_{uw} is also fixed to 1, then x_{vw} can be fixed to 1 as well. On the other hand, if x_{uw} is fixed to 0

it follows that we can also fix x_{vw} to 0. These fixings of variables are, of course, only locally valid; that is, they only apply to the current subproblem and its future descendants. A further possibility, which we have not implemented in the present code, is to fix nonbasic variables to their actual values. This can be done when the associated reduced cost of the variable exceeds the gap between the current LP value and the lower bound given by an incumbent partition.

4 Computational experiments

We have conducted extensive computational experiments with our code for the SGPP. Our first set of experiments were aimed at evaluating the usefulness of the various inequality classes within the cutting-plane algorithm, not using any branching at all. The last experiments included the full branch-and-cut algorithm with the purpose of assessing the size and range of problems that can be solved with it. This section summarizes our findings from these experiments.

The problem instances we use in the experiments are obtained from the equicut instances in [2]. We take each equicut instance with given graph structure and edge weights and create three instances of the SGPP with different cluster capacities $b = n/2$, $b = \lfloor n/4 \rfloor$, and $b = \lfloor n/6 \rfloor$, respectively. This gives a total of more than 500 SGPP instances where the associated graphs have 30 nodes or more. Some of the problem instances are highly structured in that the underlying graphs have certain grid characteristics, i.e. planar grids, toroidal grids, and mixed grids, including the set of real world instances. There are also two sets of unstructured instances — purely random instances and instances with negative and positive edge weights. See [2] for further details on the characteristics.

The computational results that are reported below were obtained by running our code for the SGPP partly on a Compaq Evo desktop computer (Pentium 4 processor, 1.5 GHz) and partly on a HP/Compaq nc6000 laptop computer (Pentium M processor, 1.6 GHz). For the LP solver we have used the dual simplex routine from the CPLEX 8.0 callable library.

4.1 Evaluation of inequality classes

Here we present some of the results from the experiments with the cutting-plane algorithm in order to get an impression of the usefulness of the various classes of facet-defining inequalities that are employed. These experiments only make use of the cutting-plane algorithm, so no branching takes place and the heuristic is not used. The cutting-plane algorithm, as described above, is modified in a number of ways during these studies: The separation procedures for some inequality classes are never called so that only the appropriate classes are used in a given experiment; the tailing-off termination mechanism is suspended so that inequality generation continues even when tailing-off occurs; and a time limit of one hour is imposed on the algorithm. This means that the cutting-plane algorithm will run until no cuts are generated or the time limit is exceeded.

In the first experiment we only use star and triangle inequalities together with the nonnegativity constraints in (1). Table 1 presents some of the problem instances where these constraints are sufficient to determine an integer optimal LP solution. Only instances that are associated with graphs on at least 40 nodes are included in this table. The first two columns in this table identify the original

equicut instance, the third and fourth columns show the number of nodes and the cluster capacity, and the last two columns give the optimal partition value and the solution time in seconds (rounded to the nearest integer). All problem instances in the table are solved well within the one-hour time limit, and it is encouraging to see that some relatively large instances are easily solved. It should be noted, however, that the solved instances in Table 1 are all associated with grid graphs.

In the second experiment we examine the effect of using S, T-inequalities and CWE inequalities in addition to the star and triangle inequalities. Table 2 presents several problem instances for which integer optimal LP solutions are obtained by using S, T inequalities and CWE inequalities. The column labelled $Z(1)$ shows the LP value when all star and triangle inequalities are satisfied, so the difference between this value and Opt. is the gap that is closed by using the additional inequalities. The columns labelled # S, T and # CWE show the numbers of S, T inequalities (including triangle inequalities) and CWE inequalities added to the LP during the optimization process. It is apparent from the table that the time consumed by the cutting-plane algorithm has increased considerably in general when compared to the times in Table 1. It is also evident that there is a strong correlation between the number of inequalities needed and the time used. Again it is encouraging to see that several relatively large problem instances can be solved with just a few inequality classes and without branching.

The next three experiments are aimed at evaluating the partial effects of using each of the classes of multistar, clique, and b-tree inequalities. Each of these inequality classes is used together with all classes of inequalities in the second experiment above, and only the still unsolved problem instances are considered. Below we present some results for the instances where one class of inequalities dominates the others; that is, the LP value obtained on termination of the cutting-plane algorithm is less than that obtained from using the other inequalities.

There are a few problem instances for which the use of multistar inequalities gives tighter bounds than using clique or b-tree inequalities. The results are shown in Table 3. Here $Z(2)$ is the bound that results from using S, T and CWE inequalities as in the second experiment, and $Z(3)$ is the final LP value obtained when multistar inequalities are also used. # MS is the total number of multistar inequalities added. There is only one of these instances for which an integer optimal LP solution is found. It should also be mentioned that the number of multistar inequalities added is sometimes quite modest in comparison with the other classes that are used. For example, in the planar grid 2×21 instance more than 23,000 CWE inequalities are added, which explains the relatively long computation time.

The clique inequalities are sometimes very helpful to obtain good bounds; in particular when b is small relative to n. Using these inequalities we have experienced the best results for mixed grid and random instances. Some of the results are shown in Table 4. Here $Z(2)$ is as above and $Z(4)$ is the final LP value that results from using the clique inequalities. # CL is the total number of clique inequalities added to the LPs. In four of these instances integer optimal LP solutions are determined within a few seconds. In most of the remaining instances considerable reductions in the gap between $Z(2)$ and Opt. are obtained with reasonable computational effort. It is remarkable that only a few clique

Type	Name	n	b	Opt.	Time (s)
mixed grid	2×20	40	10	648	0
mixed grid	2×20	40	20	884	14
mixed grid	4×10	40	20	938	9
mixed grid	5×8	40	20	938	8
mixed grid	3×14	42	21	996	16
mixed grid	6×7	42	21	996	25
mixed grid	2×22	44	22	1020	40
mixed grid	2×24	48	8	708	0
mixed grid	2×24	48	12	840	1
mixed grid	3×16	48	12	876	1
mixed grid	4×12	48	12	876	1
mixed grid	8×6	48	12	876	2
mixed grid	2×24	48	24	1164	99
mixed grid	3×16	48	24	1218	73
mixed grid	4×12	48	24	1336	65
mixed grid	6×8	48	24	1236	31
mixed grid	5×10	50	25	1320	39
mixed grid	10×6	60	30	1752	132
mixed grid	10×7	70	35	2234	373
planar grid	23×2	46	7	217	0
planar grid	6×8	48	24	355	4
toroidal grid	2×20	40	6	222	0
toroidal grid	4×10	40	20	329	3
toroidal grid	5×8	40	20	300	3
toroidal grid	8×5	40	20	293	1
toroidal grid	3×14	42	21	333	2
toroidal grid	7×6	42	21	329	3
toroidal grid	21×2	42	21	272	3
toroidal grid	12×4	48	8	350	0
toroidal grid	4×12	48	12	355	0
toroidal grid	6×8	48	12	354	1
toroidal grid	4×12	48	24	383	9
toroidal grid	12×4	48	24	412	4
toroidal grid	6×8	48	24	395	5
toroidal grid	6×10	60	30	474	23
toroidal grid	7×10	70	35	544	58
real world	mei	60	30	93	17
real world	mfi	90	15	131	7

Table 1: Some problem instances solved by using star and triangle inequalities.

Type	Name	n	b	$Z(1)$	Opt.	# S,T	# CWE	Time (s)
negative	q0 n80	40	10	236.34	234	1064	243	0
negative	q0 n60	40	20	360.58	327	7888	113	21
negative	q0 n70	40	20	326.24	313	4608	4	2
negative	q0 n80	40	20	249.50	248	1038	0	0
random	q0 90	40	6	261.03	257	747	24	0
random	q0 70	40	20	787.23	772	2806	6945	442
random	q0 90	40	20	342.07	342	2363	385	7
random	c2 90	52	26	597.03	596	6030	5177	353
random	c4 90	54	27	622.88	619	7294	4667	249
random	c6 90	56	28	691.88	691	6332	681	144
mixed grid	4×12	48	8	722.90	708	1632	422	2
mixed grid	2×25	50	25	1238.24	1230	11090	4041	1225
mixed grid	4×13	52	26	1394.00	1388	12668	4185	630
mixed grid	9×6	54	9	867.25	864	2679	194	3
mixed grid	9×6	54	27	1478.25	1476	10048	1231	385
mixed grid	10×6	60	15	1215.56	1212	5707	400	30
planar grid	2×24	48	8	236.40	233	623	374	0
planar grid	24×2	48	8	278.43	277	736	420	0
planar grid	2×24	48	12	249.80	249	1502	2287	5
planar grid	6×8	48	12	321.07	321	1626	400	1
planar grid	12×4	48	12	303.39	303	1530	800	2
planar grid	24×2	48	12	292.48	285	1426	816	1
planar grid	5×10	50	25	361.20	361	4033	1754	87
planar grid	10×5	50	25	336.44	335	4818	1231	77
planar grid	10×6	60	10	350.19	347	1893	986	2
planar grid	6×10	60	30	421.83	420	10758	4970	580
planar grid	10×6	60	30	390.21	389	9647	3126	403
planar grid	7×10	70	35	487.95	485	16505	8050	1440
toroidal grid	2×24	48	8	236.40	233	665	374	0
toroidal grid	4×12	48	8	319.50	314	1248	1212	2
toroidal grid	24×2	48	8	278.43	277	816	1044	0
toroidal grid	2×24	48	12	250.47	249	1690	2693	4
toroidal grid	24×2	48	12	292.48	286	1855	1552	4
toroidal grid	5×10	50	25	423.19	422	5550	400	25
toroidal grid	13×4	52	26	385.75	383	8620	4719	228
toroidal grid	10×6	60	15	401.12	398	5732	5781	139
toroidal grid	10×6	60	30	436.25	435	10108	2272	357
real world	mai	54	27	70.08	70	7478	5441	226
real world	mei	60	15	86.00	86	2331	800	3
real world	m6i	70	35	113.40	113	10141	2009	366
real world	mbi	74	12	104.13	103	1989	1489	8
real world	mbi	74	37	116.85	116	13571	1609	869
real world	mci	74	37	119.33	119	16419	1260	664
real world	mdi	80	20	119.23	118	10148	5627	334

Table 2: Some problem instances solved by adding S,T and CWE inequalities.

Type	Name	n	b	$Z(2)$	$Z(3)$	Opt.	# MS	Time (s)
random	t0 10	30	15	1071.01	1070.00	1070	334	560
random	t0 50	30	15	713.39	712.72	712	287	179
mixed grid	2×21	42	7	565.98	564.79	558	2565	390
mixed grid	3×14	42	7	572.36	570.00	558	984	59
mixed grid	4×11	44	11	746.41	745.20	742	367	57
mixed grid	4×13	52	13	968.93	967.66	960	1813	596
planar grid	5×8	40	6	199.80	199.21	199	549	3
planar grid	2×21	42	21	959.10	951.35	949	602	3589
toroidal grid	8×4	32	5	180.85	179.71	178	562	1

Table 3: Problem instances with best bounds from using multistar inequalities.

inequalities seem to be needed in order to enforce the bound reductions.

Like the clique inequalities, the b-tree inequalities are also very useful to obtain good bounds. Table 5 presents some of the results. Here $Z(5)$ is the LP value that is reached by the cutting-plane algorithm when b-tree inequalities are used and # b-T is the total number of these inequalities that are added. In six of these problem instances an integer optimal LP solution is obtained, and in most of the other instances an upper bound is obtained which is close to the value of an optimal partition. There are, however, three instances (negative q0 n30, mixed grid 5×8, and mixed grid 6×7) where there is still a significant gap to be closed. We note that relatively many b-tree inequalities are used to obtain the bounds and that the cutting-plane algorithm runs through many iterations in these cases, which is why the time spent by the algorithm often extends over several minutes.

The above results suggest that the classes of S, T, CWE, clique, and b-tree inequalities are the most useful of the five inequality classes that are considered in this paper. Since, however, the multistar inequalities can be useful in some instances we have chosen also to use them in our branch-and-cut algorithm. We remark that our conclusion is based on the particular separation procedures we have implemented for these inequality classes — better separation procedures for some classes may affect the outcome — so this conclusion is not a definite one.

4.2 Branch-and-cut results

Here we present some computational results of the branch-and-cut algorithm. For each type of problem we consider a few of the larger instances with different cluster capacities in order to demonstrate the size and range of problem instances that can be solved. In these experiments we have allowed the branch-and-cut algorithm to run for a maximum of five hours. This means that some problem instances have not been solved to proven optimality. Table 6 shows the results. The column labelled Heur shows the value of the best partition found by the heuristic. Z_{LP} is the final LP value obtained at the root of the branch-and-cut enumeration tree. Branch is the number of subproblems considered during the enumeration; this includes the root problem so that a 1 in this column means that the problem instance is solved without branching. Since all edge weights are integers, a branch in the enumeration tree is pruned whenever the

Type	Name	n	b	$Z(2)$	$Z(4)$	Opt.	# CL	Time (s)
random	q0 00	40	6	718.45	700.66	687	3	21
random	q0 10	40	6	693.33	677.52	662	3	26
random	q0 20	40	6	664.19	650.01	632	2	25
random	q0 30	40	6	629.45	617.72	605	2	25
random	q0 40	40	6	589.56	579.95	567	2	21
random	q0 60	40	6	495.25	488.57	483	3	14
random	q0 70	40	6	426.29	422.48	416	2	9
random	q0 80	40	6	355.05	353.44	350	2	5
random	c0 70	50	8	674.34	672.81	666	2	130
random	c2 90	52	13	501.30	498.79	496	8	64
mixed grid	2×19	38	9	573.87	567.32	550	53	135
mixed grid	2×20	40	6	513.69	510.00	510	19	0
mixed grid	4×10	40	6	510.24	505.15	501	14	0
mixed grid	5×8	40	6	514.00	510.00	510	5	0
mixed grid	2×21	42	10	666.00	658.00	658	31	4
mixed grid	3×14	42	10	686.15	677.55	558	45	152
mixed grid	6×7	42	10	691.13	681.47	658	1	16
mixed grid	2×22	44	7	590.70	585.91	576	73	94
mixed grid	4×11	44	7	599.97	594.58	576	7	3
mixed grid	2×23	46	7	615.00	609.00	606	68	149
mixed grid	2×25	50	8	724.00	718.00	718	47	1
mixed grid	5×10	50	12	893.07	882.01	868	9	27
mixed grid	4×13	52	8	767.00	753.00	746	23	2

Table 4: Some problem instances with best bounds from using clique inequalities.

LP value exceeds the incumbent partition value by less than one. This is why no branching takes place when $Z_{\text{LP}} < \text{Heur} + 1$.

The *negative and random problem instances* have varying densities in terms of the number of nonzero edge weights. Our experience with these instances shows that relatively dense instances are hard to solve, while the very sparse instances are more easy to solve. The last two digits in the problem name give the percentage of zero weights; for example, negative c0 n30 is relatively dense in that only 30% of the edges have zero weights. We conclude that dense instances with 50 nodes or more cannot be solved within a reasonable time limit.

Despite the fact that some of the *mixed grid instances* are easily solved (cf. Tables 1 and 2), some of the other instances appear to be very hard to solve. For example, two of the instances in Table 6 cannot be solved within the 5-hour time limit. The reason is that these problem instances are very regular in structure — all edges in the grid have weight 10 and all other edges have weight 1 — so that there exist several alternative optimal solutions. Sometimes the gap between Z_{LP} and the optimal partition value is large, and it may be extremely difficult to close this gap during the branch-and-cut enumeration.

We have been able to solve all the *planar grid instances* in the test set. Indeed, most of these instances are easily solved without branching when $b = n/2$ (a few examples are given in Tables 1 and 2). Table 6 gives some examples with smaller capacities b. All *toroidal grid instances* have also been solved.

Type	Name	n	b	$Z(2)$	$Z(5)$	Opt.	# b-T	Time (s)
negative	q0 n30	40	6	364.17	363.15	351	4778	80
negative	q0 n60	40	6	298.32	297.23	297	4486	62
negative	q0 n70	40	6	281.16	279.33	278	6964	210
negative	q0 n70	40	10	305.47	305.00	305	2443	71
negative	c0 n80	50	8	308.23	308.00	308	10329	1542
random	q0 90	40	10	292.06	291.09	288	2329	64
random	c0 90	50	8	403.65	402.51	402	7867	379
mixed grid	5×8	40	10	656.76	655.68	648	4994	364
mixed grid	6×7	42	7	577.23	575.77	558	6125	205
mixed grid	6×8	48	8	714.00	711.87	708	9504	910
planar grid	4×10	40	6	224.18	224.00	224	2685	27
planar grid	8×5	40	6	216.22	216.00	216	2481	11
planar grid	10×4	40	6	205.25	203.79	203	5399	97
planar grid	4×12	48	8	281.14	281.00	281	465	0
toroidal grid	4×10	40	6	240.72	238.62	237	2220	21
toroidal grid	3×14	42	7	272.09	270.91	270	7307	319
toroidal grid	7×6	42	7	267.28	265.66	265	6318	210
toroidal grid	4×11	44	11	323.37	323.00	323	1350	67
toroidal grid	6×8	48	8	316.91	315.82	314	6689	389

Table 5: Some problem instances with best bounds from using b-tree inequalities.

However, these instances seem to be somewhat harder to solve in general than the corresponding planar grid instances. The examples given in the table also indicate this; note that one instance is only barely solved within the 5-hour time limit.

The set of *real world problems* contains the largest instances we have attempted to solve. The largest instance on 148 nodes is only solved for $b = 74$, in which case it is easily solved. For the smaller cluster capacities the algorithm is unable to solve the problem; in the case with $b = 37$ the cutting-plane algorithm does not complete its processing of the root problem within the time limit. Most of the other problem instances in this set are solved without branching as can be seen in Table 6.

5 Concluding remarks

The branch-and-cut algorithm presented in this paper works well on most problem instances we have attempted to solve. In particular, this is true with regard to instances that are associated with grid graphs on 100 or fewer nodes. With regard to the unstructured instances, however, the densities of the graphs to be partitioned seem to be the single most deciding factor for the solvability of the problems. In general, sparse problem instances are easily solved, while dense ones are hard to solve — the latter instances may not be solved if the graph has 50 nodes or more.

The cluster capacity b may also influence the ability to solve a problem. More than half of the instances that are easily solved without branching have capacity $b = n/2$. Branching takes place more frequently when the capacities

Type	Name	n	b	Heur	Z_{LP}	Opt.	Branch	Time (s)
negative	c0 n30	50	8	518	552.49	n.a.	n.a.	18000
negative	c0 n40	50	8	556	571.55	556	95	2361
negative	c0 n50	50	8	504	522.11	504	157	3424
negative	c0 n30	50	12	561	590.98	561	361	13092
negative	c0 n40	50	12	571	600.88	571	301	8824
negative	c0 n50	50	12	536	564.19	542	93	3354
negative	c0 n30	50	25	582	593.26	582	33	1244
negative	c0 n40	50	25	575	599.97	577	115	3649
negative	c0 n50	50	25	549	570.34	549	95	2736
random	c0 70	50	8	666	674.29	666	27	813
random	c0 80	50	8	550	560.20	555	37	1565
random	c0 90	50	8	402	402.98	402	1	31
random	c0 70	50	12	792	828.23	n.a.	n.a.	18000
random	c0 80	50	12	646	662.61	648	341	14651
random	c0 90	50	12	454	454.57	454	1	25
random	c0 70	50	25	1189	1233.32	1189	31	8539
random	c0 80	50	25	887	896.41	887	13	2218
random	c0 90	50	25	550	550.00	550	1	10
mixed grid	2×23	46	7	606	609.00	606	141	6194
mixed grid	2×23	46	11	734	752.00	n.a.	n.a.	18000
mixed grid	2×23	46	23	1082	1084.15	1082	3	414
mixed grid	5×10	50	8	709	729.41	n.a.	n.a.	18000
mixed grid	5×10	50	12	868	882.02	868	221	7490
mixed grid	4×13	52	8	746	753.00	746	629	4058
mixed grid	4×13	52	13	960	967.95	960	19	2341
planar grid	5×10	50	8	307	308.36	308	3	58
planar grid	5×10	50	12	333	334.18	333	3	110
planar grid	6×10	60	10	362	366.55	365	7	495
planar grid	6×10	60	15	388	392.31	389	19	4010
planar grid	7×10	70	11	433	434.37	434	3	376
planar grid	7×10	70	17	460	460.60	460	1	47
toroidal grid	13×4	52	8	319	319.81	319	1	2
toroidal grid	13×4	52	13	352	352.91	352	1	428
toroidal grid	6×10	60	10	386	390.24	386	33	1276
toroidal grid	6×10	60	15	419	420.65	420	3	168
toroidal grid	7×10	70	11	459	464.42	462	63	5505
toroidal grid	7×10	70	17	491	496.14	491	101	17616
real world	mdi	80	13	110	110.83	110	1	40
real world	mdi	80	40	125	125.77	125	1	6
real world	m1i	100	16	140	140.91	140	1	2
real world	m1i	100	25	145	145.97	145	1	221
real world	m1i	100	50	152	152.42	152	1	6
real world	m8i	148	24	230	231.66	n.a.	n.a.	18000
real world	m8i	148	37	241	n.a.	n.a.	n.a.	18000
real world	m8i	148	74	258	258.61	258	1	84

Table 6: Some branch-and-cut results.

are smaller, but we cannot conlude that instances with small capacities are consistently harder to solve than instances with larger capacities.

The fact that extensive branching sometimes takes place indicates that more classes of facet-defining inequalities may be needed in order to solve some of the hard problem instances. Whether some of the already known inequality classes from the literature can be useful in a branch-and-cut framework like the present one is an open question that can only be answered through further computational studies. Another important research direction is the search for new classes of facets of $P_n(b)$ and related polytopes.

References

[1] F. Barahona, M. Jünger, G. Reinelt, Experiments in quadratic 0–1 programming, Mathematical Programming 44 (1989) 127–137.

[2] L. Brunetta, M. Conforti, G. Rinaldi, A branch-and-cut algorithm for the equicut problem, Mathematical Programming 78 (1997) 243–263.

[3] U. Faigle, R. Schrader, R. Suletzki, A cutting plane algorithm for optimal graph partitioning, Methods of Operations Research 57 (1986) 109–116.

[4] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, L.A. Wolsey, Formulations and valid inequalities for the node capacitated graph partitioning problem, Mathematical Programming 74 (1996) 247–266.

[5] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, L.A. Wolsey, The node capacitated graph partitioning problem: A computational study, Mathematical Programming 81 (1998) 229-256.

[6] M. Grötschel, Y. Wakabayashi, A cutting plane algorithm for a clustering problem, Mathematical Programming 45 (1989) 56–96.

[7] M. Grötshcel, Y. Wakabayashi, Facets of the clique partitioning polytope, Mathematical Programming 47 (1990) 367–387.

[8] S. Holm, M.M. Sørensen, The optimal graph partitioning problem, OR Spektrum 15 (1993) 1–8.

[9] E.L. Johnson, A. Mehrotra, G.L. Nemhauser, Min-cut clustering, Mathematical Programming 62 (1993) 133–151.

[10] E.M. Macambira, C.C. de Souza, The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations, European Journal of Operational Research 123 (2000) 346–371.

[11] M.M. Sørensen, An adaptation of the Kernighan-Lin heuristic to the simple graph partitioning problem, Working paper 99-1, Dept. of Management Science and Logistics, Aarhus School of Business, 1999. (Available at www.hha.dk/∼mim/papers.htm.)

[12] M.M. Sørensen, Facet defining inequalities for the simple graph partitioning polytope, Working paper 00-3, Dept. of Management Science and Logistics, Aarhus School of Business, 2000. (Revised version available at www.hha.dk/∼mim/papers.htm.)

[13] M.M. Sørensen, New facets and a branch-and-cut algorithm for the weighted clique problem, European Journal of Operational Research 154 (2004) 57–70.

[14] M.M. Sørensen, b-Tree facets for the simple graph partitioning polytope, Journal of Combinatorial Optimization 8 (2004) 151–170.

[15] C.C. de Souza, M. Laurent, Some new classes of facets for the equicut polytope, Discrete Applied Mathematics 62 (1995) 167–191.

Working Papers from Logistics/SCM Research Group

L-2005-02 Michael M. Sørensen: Polyhedral computations for the simple graph partitioning problem.

L-2005-01 Ole Mortensen: Transportkoncepter og IT-støtte: et undersøgelsesoplæg og nogle foreløbige resultater.

L-2004-05 Lars Relund Nielsen, Daniele Pretolani & Kim Allan Andersen: *K* shortest paths in stochastic time-dependent networks.

L-2004-04 Lars Relund Nielsen, Daniele Pretolani & Kim Allan Andersen: Finding the *K* shortest hyperpaths using reoptimization.

L-2004-03 Søren Glud Johansen & Anders Thorstenson: The (*r*,*q*) policy for the lost-sales inventory system when more than one order may be outstanding.

L-2004-02 Erland Hejn Nielsen: Streams of events and performance of queuing systems: The basic anatomy of arrival/departure processes, when focus is set on autocorrelation.

L-2004-01 Jens Lysgaard: Reachability cuts for the vehicle routing problem with time windows.

ISBN 87-7882-080-4

Department of Accounting, Finance and Logistics
Faculty of Business Administration

Aarhus School of Business
Fuglesangs Allé 4
DK-8210 Aarhus V - Denmark

Tel. +45 89 48 66 88
Fax +45 86 15 01 88

www.asb.dk