Algorithmic Aspects of Using Small Instance Relaxations in Parallel Branch-and-Cut

Thomas Christof and Gerhard Reinelt
Institut für Angewandte Mathematik
Universität Heidelberg
Im Neuenheimer Feld 293/294
D-69120 Heidelberg, Germany
Gerhard.Reinelt@IWR.Uni-Heidelberg.De

April 1, 1998

Abstract

Essential for the success of branch-and-cut algorithms for solving combinatorial optimization problems are the availability of reasonable tight relaxations and effective routines for solving the associated separation problems. In this paper we introduce the concept of small instance relaxations which can be particularly useful for problems with symmetric structure. Small instance relaxations base on the facets of polytopes associated with small instances of the combinatorial optimization problem to be solved and can be generated automatically by facet enumeration. For a certain class of symmetric problems, we describe a general approach to the separation problem. Algorithmic aspects of using small instance relaxations effectively (parallel separation, facet selection, cutting plane selection) are discussed in detail. Extensive computational results are presented for the linear ordering problem and a certain betweenness problem.

1 Introduction

During the last years, branch-and-cut algorithms have become in many cases the method of choice when trying to solve hard combinatorial optimization problems. Whereas the principle idea is easily understood, its application to a particular optimization problem and the design and implementation of corresponding software are nontrivial. Meanwhile, due to the availability of software frameworks like ABACUS [Thi95, JT97a, JT97b] or MINTO [NSS94], implementation has been facilitated and many general components of branch-and-cut algorithms do not have to be re-implemented when attempting to solve some combinatorial optimization problem.

However, what is still problem dependent and an ambitions mathematical task, is the theoretical investigation of polytopes that are associated with a problem. The facial structure of the polytopes has to be examined, and even more important, effective separation algorithms for classes of inequalities belonging to the facial description have to be designed and implemented. The availability of effective separation routines decides whether classes of inequalities are interesting and useful for practical computations or not. It is often the case that from a wealth of theoretical knowledge for some optimization problem only little can actually be exploited in practice.

It is the purpose of this paper to show that some of this gap between theory and practice can be closed. Namely, by focusing on small instances of problems already a lot of insight into the facial structure of associated polytopes can be gained. Of particular interest is that this insight can be obtained with the help of computer by applying powerful facet enumeration algorithms. Based on small instances we introduce the concept of small instance relaxations and show that for these relaxations the separation problem can be handled with a general approach and is very well suited for parallelization. With extensive practical tests we explore how small instance relaxations can be used most effectively within branch-and-cut algorithms and show that they can improve problem solving capability substantially.

The paper is organized as follows. In section 2 we give a short introduction to the branch-and-cut approach. Section 3 motivates the use of small instance relaxations and describes a general separation approach. In section 4 we address parallelization of cut generation. Possible strategies are outlined in section 5. Section 6 and 7 present detailed computational experiments for the linear ordering problem and a weighted betweenness problem. A final discussion in section 8 concludes this paper.

2 Combinatorial Optimization and Branch-and-Cut

Many combinatorial optimization problems belong to the following general problem class. Let E be a finite set (ground set) and $\mathcal{I}$ be a set of subsets of E (feasible solutions). For every $e \in E$ there is a weight $c(e)$ (objective function) and the objective function value of a feasible solution $I \in \mathcal{I}$ is defined as

$$c(I) := \sum_{e \in I} c(e).$$

The optimization problem then consists of finding $I^* \in \mathcal{I}$ such that $c(I^*) = \max\{c(I) \mid I \in \mathcal{I}\}$, or in the case of a minimization problem $c(I^*) = \min\{c(I) \mid I \in \mathcal{I}\}$.

Among many approaches to solving this type of problem the so-called *branch-and-cut* approach turned out to be very successful. This approach basically works as follows. With every $I \in \mathcal{I}$ we associate its $|E|$-dimensional *characteristic* or *incidence vector* χ^I defined by setting

$$\chi_e^I := \begin{cases} 1 & \text{if} \quad e \in I, \\ 0 & \text{if} \quad e \notin I. \end{cases}$$

For given $\mathcal{I}$ we define the polytope

$$P_{\mathcal{I}} = \mathrm{conv}(\{\chi^I \mid I \in \mathcal{I}\}).$$

The vertices of this polytope correspond exactly to the feasible characteristic vectors, i.e., x is a vertex of $P_{\mathcal{I}}$ if and only if $x = \chi^I$ for some $I \in \mathcal{I}$.

Having a description of $P_{\mathcal{I}}$ with linear inequalities, e.g. $P_{\mathcal{I}} = \{x \mid \bar{A}x \leq \bar{b}\}$, then the combinatorial optimization problem can be solved by finding an optimum vertex solution of the linear programming problem

$$\begin{aligned} &\max c^T x \\ &\bar{A}x \leq \bar{b}. \end{aligned}$$

Whereas the existence of finite systems $\bar{A}x \leq \bar{b}$ follows from results of polyhedral theory, there is no hope for determining this linear description for hard combinatorial optimization problems explicitly. One has to be satisfied with relaxations

$$P_{\mathcal{I}} \subseteq \{x \mid Ax \leq b\}.$$

Therefore, when now solving

$$\begin{aligned} &\max c^T x \\ &Ax \leq b \end{aligned}$$

it is possible that an optimal solution is not a feasible 0/1 solution. In such a case one has to resort to branch-and-bound.

There are several difficulties with this principle approach (for a detailed discussion we refer to [JRT95]). The main difficulty comes from the fact that although $Ax \leq b$ is only a partial system it usually contains a number of inequalities exponential in $|E|$. So it is not possible to list these inequalities explicitly. Here the cutting part of branch-and-cut comes into play. Namely one starts the optimization of $\max\{c^T x \mid Ax \leq b\}$ with a subsystem of $Ax \leq b$, and after solving the current linear program checks if all other inequalities are satisfied. If not, then inequalities are added, and the process is iterated. Now it is clear that this approach can only work if violated inequalities can be identified effectively. This is the so-called *separation problem*.

For obtaining an effective branch-and-cut algorithm for a combinatorial optimization problem given by E, $\mathcal{I}$, and c we therefore need two prerequisites. First, we need a good partial description

$$P_{\mathcal{I}} \subseteq \{x \mid Ax \leq b\}$$

approximating $P_{\mathcal{I}}$ as closely as possible. Usually we require that an inequality $a^T x \leq b$ of $Ax \leq b$ is *facet-defining*, meaning that the induced face $F = P_{\mathcal{I}} \cap \{x \mid a^T x = b\}$ is of maximal dimension $\dim P - 1$. Second, we have to have procedures available that identify violated inequalities quickly. The latter point actually decides whether a theoretical result about a subsystem of inequalities needed for describing $P_{\mathcal{I}}$ is useful for practical computations or not. In the following we will therefore concentrate on finding partial linear descriptions $P_{\mathcal{I}} \subseteq \{x \mid Ax \leq b\}$ with inequalities that can be separated effectively.

Note that now it is not guaranteed that a 0-1 vector satisfying $Ax \leq b$ corresponds to a feasible solutions. Usually a system $Ax \leq b$ such that $\{x \in \{0,1\}^{|E|} \mid Ax \leq b\} = \{\chi^I \mid I \in \mathcal{I}\}$ is known. However, if this is not the case, some care has to be given to infeasible 0/1-vectors.

Furthermore, note that some inequalities of $Ax \leq b$ can actually be equations. The number of independent equations determines the dimension of $P_{\mathcal{I}}$ and is comparatively small (at most $|E|$).

3 Small Instance Relaxations

Whereas classes of inequalities belonging to a partial description and corresponding separation routines are mostly found by theoretical investigations we will concentrate on a completely different approach in this paper.

For small problem instances of combinatorial optimization problems, usually the set of feasible incidence vectors corresponding to the set of feasible solution can be enumerated. By then applying algorithms for facet enumeration, the linear description of $P_{\mathcal{I}}$ can be found in a standardized way. We do not elaborate on the technical difficulities that are involved here, but refer to [CR97a].

Such descriptions can be the basis for theoretical research on the polyhedral structure of $P_{\mathcal{I}}$. We are interested in such descriptions for the following reason. When going from small instances to larger instances, facet-defining inequalities for a small instance can be extended (*lifted*) to facet-defining inequalities for the larger instance by computing suitable coefficients for the newly introduced variables (and possibly a new right hand side). In general, it is difficult to compute these coefficients, but often already an extension with zero coefficients preserves the facet-defining property (*trivial lifting*). Such inequalities are particularly useful for our approach.

By *small instance relaxation (SIR)* of a linear combinatorial optimization problem we denote a relaxation which is obtained by lifting of facet-defining inequalities of low-dimensional 0/1-polytopes associated with small instances of the problem. Note that this notion reflects more a general idea than an exact definition. The difference to traditional approaches is that we want to use automatically generated inequalities without studying and exploiting specific knowledge about the structure of the coefficients forming the inequality.

Small instance relaxations can be extremely useful if the problem meets two requirements:

- In order to use small instance relaxations, the structure of the polytopes of small instances has to be related to the corresponding polytope of any size. Given such a relationship, we assume that this relationship can be expressed in terms of lifting rules.

- The polytope should be symmetric in some sense such that the vertices and facets can be assigned to equivalence classes. Then precomputed lists of facets of the small instances (each representing a single class) can be used in branch-and-cut algorithms. Symmetry and lifting operations produce a large set of valid inequalities which can be considered for separation.

Since we do not require a certain kind of symmetry at this point, the idea of small instance relaxations is quite general. To specify a particular symmetry structure in detail, we now introduce the concept of *σ-symmetry* which covers a large class of well-known combinatorial optimization problems.

Definition 3.1 (σ-symmetry) *Let a combinatorial optimization problem be given where the elements of the ground set correspond to the edges of a complete (directed or undirected) graph with n nodes. We call the problem σ-symmetric if for a feasible solution I and any permutation σ of $\{1, \ldots, n\}$ also $\sigma(I)$ (obtained by transforming an edge (i, j) to $(\sigma(i), \sigma(j))$) is a feasible solution. In this case we say that the associated polytope is σ-symmetric.*

Many interesting combinatorial optimization problems are σ-symmetric. Examples are the traveling salesman problem, the linear ordering problem, or the max-cut problem.

Let $P = \text{conv}(X) \subseteq \mathbb{R}^d$ be a σ-symmetric 0/1-polytope where X is the set of incidence vectors of all feasible solutions. Then we have that for an incidence vector $\chi \in X$ and every permutation σ of the nodes the vector $\sigma(\chi)$ defined by setting $\sigma(\chi)_{ij} = \chi_{\sigma(i)\sigma(j)}$, for all $1 \leq i, j \leq n$, is also a feasible incidence vector.

In this paper we restrict ourselves to σ-symmetric combinatorial optimization problems.

We now discuss the separation problem for small instance relaxations associated with σ-symmetric combinatorial optimization problem. If trivial lifting is possible, then the separation problem for a σ-symmetric polytope can be transformed to a quadratic assignment problem. The quadratic assignment problem itself has been studied thoroughly and can be successfully attacked by heuristics. The important observation is that this approach for solving the separation problem works for all σ-symmetric combinatorial optimization problems.

Definition 3.2 (Quadratic assignment problem) *Let a set $N = \{1, \ldots, n\}$ and matrices $U = (u_{ij})$ and $V = (v_{kl})$, $U, V \in \mathbb{R}^{n \times n}$ be given. The quadratic assignment problem (QAP) can be stated as follows:*

$$\min_{\pi \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij} v_{\pi(i)\pi(j)},$$

where Π_N is the set of all permutations of N.

The QAP is an $\mathcal{NP}$-hard combinatorial optimization problem. A survey is [PRW94], recent polyhedral approaches are [PR96, Kai97].

Let x^* be a fractional LP solution connected with the complete directed graph $D_n = (V_n, A_n)$ with n nodes and let $g^T x \leq g_0$ be an inequality of a small instance relaxation obtained by trivial lifting of an inequality $f^T x \leq f_0$ defined on a graph with $\overline{n} < n$ nodes. Then define

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x^*_{ij} g_{\pi(i)\pi(j)}.$$

The set

$$\Pi^*_N = \{\pi \in \Pi_N \mid C(\pi) > g_0\}$$

induces the set of different violated inequalities $\pi(g)^T x \leq g_0$ associated with the facet class $f^T x \leq f_0$. (If the graph is undirected we can transform to the directed case by setting $x^*_{ji} = x^*_{ij}$, for $j > i$, and $f_{ji} = g_{ji} = 0$, for $j > i$.)

Finding Π^*_N or a subset of Π^*_N requires to find feasible solutions to the QAP

$$\min_{\pi \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} -x^*_{ij} g_{\pi(i)\pi(j)}$$

with objective function value $-C(\pi)$ less than $-g_0$.

Since $g^T x \leq g_0$ is obtained from $f^T x \leq f_0$ by trivial lifting, it holds that $g_{ij} = 0$ for $i > \overline{n}$ and $j > \overline{n}$. For every permutation π we have

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x^*_{ij} g_{\pi(i)\pi(j)} = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x^*_{\pi^{-1}(i)\pi^{-1}(j)} g_{ij} = \sum_{i=1}^{\overline{n}} \sum_{j=1, j \neq i}^{\overline{n}} x^*_{\pi^{-1}(i)\pi^{-1}(j)} f_{ij}.$$

Therefore it is sufficient to find suitable settings of $\pi^{-1}(i)$, for $1 \leq i \leq \overline{n}$. Note that usually n can be considerably larger than $\overline{n}$.

As the separation problem in a branch-and-cut algorithm has to be solved many times and as we are not required to find the global optimum of the QAP, but a solution with a value less than a certain bound, it is favourable to use a heuristic.

There is a wide range of heuristics for the QAP. Since our problems do not exploit any structure of the underlying facet-defining inequality we selected a Grasp heuristic [LPR94] which has been reported to be quite effective [PRW94]. Grasp (Greedy Randomized Adaptive Search Procedure) is an iterative process, with each iteration consisting of two phases, a *construction phase* and a *local search phase*. The best overall solution is kept as a heuristic solution to the QAP. Algorithm 3.3 gives the principle of a Grasp heuristic.

Algorithm 3.3 (Grasp)

(1) Initialization phase.

(2) For a given number of iterations r_{gr}

(2.1) Construction phase.

(2.2) Local search phase.

For our application we have realized the Grasp approach as follows.

In the initialization phase (Algorithm 3.4) a couple of promising assignments is built in a greedy fashion which is then used in step (1) of the construction phase (Algorithm 3.5) for the first 2 assignments of the permutation to be constructed. The following assignments are made in a randomized greedy fashion in step (2) of the construction phase so that in each iteration the score c_{ik} with respect to the already made assignments is maximized.

Algorithm 3.4 (Initialization phase)

(1) Let $s(f)$ be the number of non-zero coefficients of f. Sort the non-zero coefficients of f in decreasing order

$$f_{i_1 j_1} \geq f_{i_2 j_2} \geq \ldots f_{i_{s(f)} j_{s(f)}}.$$

(2) Let $s(x^*)$ be the number of non-zero entries of x^*. Sort the values of the LP solution x^* in decreasing order

$$x^*_{k_1 l_1} \geq x^*_{k_2 l_2} \geq \ldots x^*_{k_{s(x^*)} l_{s(x^*)}}.$$

(3) Keep the $s(f)$ largest products

$$x^*_{k_1 l_1} f_{i_1 j_1} \geq x^*_{k_2 l_2} f_{i_2 j_2} \geq \ldots x^*_{k_{s(f)} l_{s(f)}} f_{i_{s(f)} j_{s(f)}}.$$

In the following $0 < \alpha < 1$ is a candidate restriction parameter.

Algorithm 3.5 (Construction phase)

*(1) Generate a random integer r between 1 and $\lfloor \alpha s(f) \rfloor$ and use the $r-th$ product $x^*_{k_r l_r} f_{i_r j_r}$ for the first two assignments $\pi^{-1}(i_r) = k_r$ and $\pi^{-1}(j_r) = l_r$.
Initialize a list $\mathcal{L}$ of assignments as $\mathcal{L} = \{(i_r, k_r), (j_r, l_r)\}$.*

(2) For assignments 3 to $\overline{n}$ do:

(2.1) Setup an empty heap.

(2.2) For all possible assignments (i, k), $1 \leq i \leq \overline{n}$, $1 \leq k \leq n$, insert the assignment (i, k) with score

$$c_{ik} = \sum_{(j,l) \in \mathcal{L}} (x^*_{kl} f_{ij} + x^*_{lk} f_{ji})$$

*into the heap.
(An assignment (i, k) is "possible" if there exists no assignments $(i, .)$ and $(., k)$ in $\mathcal{L}$.)*

(2.3) Let n_h be the number of elements in the heap. Choose a random number r between 1 and $\lfloor \alpha n_h \rfloor$. Extract r times the maximum element of the heap, let c_{ik} be the last element. Set $\mathcal{L} = \mathcal{L} \cup \{(i, k)\}$.

(3) Construct a permutation π from $\mathcal{L}$ with $\pi^{-1}(i) = k$ for $(i,k) \in \mathcal{L}$.

The permutations generated by this construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed permutation. The most commonly used neighborhood structure for QAP is the 2-exchange neighborhood structure

$$N_2(\pi) = \{\rho \mid d(\rho,\pi) \leq 2\}, \text{where } d(\pi,\rho) = |\delta(\pi,\rho))| \text{ and } \delta(\rho,\pi) = \{i \mid \pi(i) \neq \rho(i)\}.$$

Define the set X of all possible *pairwise exchanges* as

$$X = \{(i,j) \mid i \neq j, 1 \leq i,j \leq n\}.$$

For a given permutation π, denote by $N_2(\pi, X')$ the set of permutations in $N_2(\pi)$ obtained by performing pairwise exchanges in a subset of pairwise exchanges $X' \subseteq X$.

An application of the local search procedure of [LPR94] for our purposes is the following.

Algorithm 3.6 (Local search phase)

(1) Set a boolean variable `globalBetter=true`.

(2) While `globalBetter=true`

- *(2.1) Set $\pi_0 = \pi$, $X_0 = X$, $k = 0$, set a boolean variable* `localBetter=true`.
- *(2.2) While* `localBetter=true`
  - *(2.2.1) Find the best permutation $\pi_{k+1} \in N_2(\pi_k, X_k)$.*
  - *(2.2.2) If $\pi_{k+1} \in \Pi_N^*$, i.e., the current objective function value $C(\pi_{k+1})$ exceeds the bound given by the right hand side of the inequality, generate a cutting plane.*
  - *(2.2.3) Set $X_{k+1} = X_k \setminus \{(i,j) \mid i \text{ and } j \in \cup_{l=0}^{k} \delta(\pi_l, \pi_{l+1}), i,j = 1,\ldots,n\}$.*
  - *(2.2.4) If $C(\pi_{k+1}) \leq C(\pi_0)$, then set* `localBetter=false`.
  - *(2.2.5) Set $k = k+1$.*
- *(2.3) Set $\pi = \text{argmax}\{C(\pi_1),\ldots,C(\pi_k)\}$*
- *(2.4) If $C(\pi) = C(\pi_0)$, then set* `globalBetter=false`.

As long as the objective function value can be improved (`globalBetter=true`) a local search step (step (2)) is executed. At each local search step, the current assignment is denoted by π_0 and the set X_0 of allowed pairwise exchanges is initialized as the set of all possible pairwise exchanges. In such a search step permutations $\pi_1,\ldots,\pi_k$ in the neighborhood are constructed in an iterative fashion. The neighborhood structure itself varies as the set X_{k+1} is updated. Hereby X_{k+1} is derived from X_k (step (2.2.3)) by excluding certain pairwise exchanges. In step (2.3) among all permutations $\pi_1,\ldots,\pi_k$ the best one is chosen (step (2.3)) as the new permutation π.

We will report on the performance of the Grasp heuristics below.

4 Parallel Cutting Plane Generation

There is one possible drawback in using small instance relaxations that we have not discussed so far. The number of different facet classes even for small instances is enormous, for example at least 15379 classes for the traveling salesman polytope on 10 nodes or at least 12231 classes for the linear ordering polytope on 8 nodes [CR97a]. Morevover, these classes have various properties which may have impact on their importance for practical computations.

Before addressing the usefulness of facet classes in the subsequent sections, we describe a software and hardware setup that allows for an effective use of separation routines for small instance relaxations.

First research on parallel cutting plane generation is reported in [CR95, CR96]. In these papers, it was shown that for the traveling salesman problem and for the linear ordering problem parallel cutting plane generation in combination with the use of small instance relaxations for separation is a promising new approach to parallel branch-and-cut. We now refine the concepts of parallel cutting plane generation.

The special structure of the separation problem connected with small instance relaxations suggests to execute the separation procedure in parallel, because all classes of facets can be treated simultaneously. Therefore, we can distribute the separation heuristics among the available processors of a parallel hardware. In the following setup for a parallel branch-and-cut code, we solely concentrate on parallelizing the generation of cutting planes and neglect other possibilities for parallelization. We discuss the parallel approach using the example of the separation problem connected with small instance relaxations, but our parallel approach can be adapted without any difficulties to be used for any kind of specialized separation algorithms.

The general idea is as follows (see also Figure 1). Let N be the number of available processors. The core of the branch-and-cut algorithm (including standard separation heuristics) runs on processor P_1. This corresponds basically to a sequential implementation. Fractional solutions of LP relaxations are sent to the other processors. The task of every processor P_i, $2 \leq i \leq N$, is to find cutting planes which are then passed to processor P_1. Processor P_1 collects the inequalities found by the other processors and whenever the standard separation algorithms cannot produce enough cutting planes, then these violated inequalities are added to the system of active constraints.

By means of a local pool at every processor P_i, $2 \leq i \leq N$, for storing all violated inequalities, cutting planes can be generated in further separation phases also by pool separation. (We assume that all inequalities are globally valid, i.e., even if we have to generate branching nodes, all inequalities are valid for every node of the branching tree.)

If every processor executes different separation algorithms, then the same cutting plane cannot be generated by two different processors. In our application of SIR-separation, each one of these processors is responsible for a certain subset of the facet classes of the small instance relaxation to be separated. For every class of inequalities that a processor is responsible for, it executes the Grasp heuristics for separation.

Algorithms 4.1 and 4.2 give more detailed descriptions of the control flows of processor P_1 and processors P_i, $2 \leq i \leq N$. Parallel cutting plane generation can be organized in *synchronous*, or alternatively in *asynchronous* mode. In the latter case, the processors P_i, $2 \leq i \leq N$, generate cutting planes for storing them in the local pool, even if processor P_1 at the same time reoptimizes the linear program. In both modes, first pool separation is performed by processors P_i, $2 \leq i \leq N$, followed by SIR-separation. The parameter max_{snd} is the maximal number of cutting planes sent by a single processor P_i, $2 \leq i \leq N$, in one separation phase. In asynchronous mode, the additional parameter max_{gen}, $max_{gen} > max_{snd}$, is the total number of cutting planes generated by a single processor P_i, $2 \leq i \leq N$, in one separation phase.

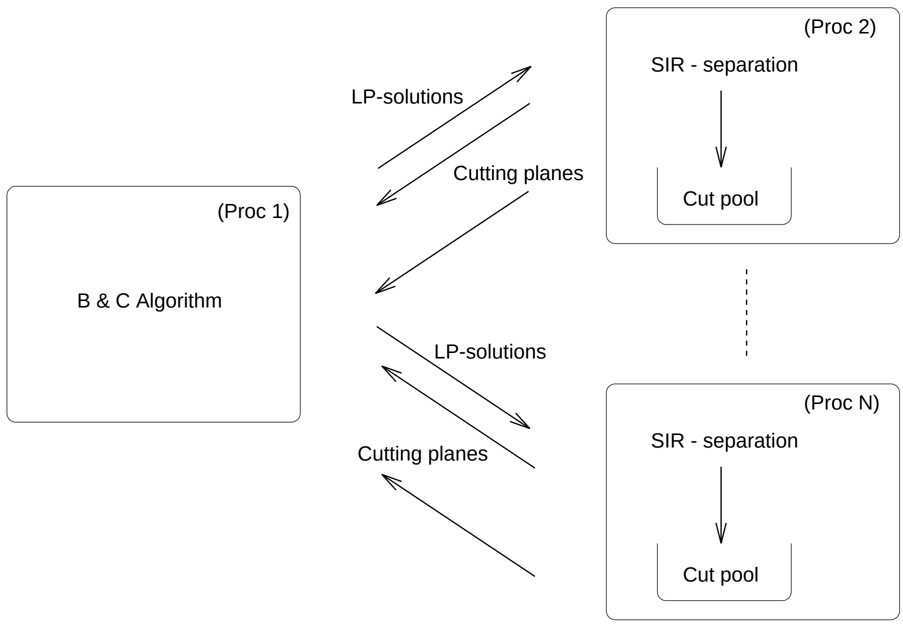

Figure 1: Parallel cutting plane generation.

Algorithm 4.1 (Parallel branch-and-cut (processor P_1))

(1) Run the sequential branch-and-cut code modified by executing the following statements in every separation phase.

(1.1) Execute standard separation routines.

(1.2) If the number of cutting planes found by the standard separation routines is less than a certain number t_1, send the current fractional LP solution x_k^* to every processor P_i, $2 \leq i \leq N$.

(1.3) If `mode=synchronous`*, then*

(1.3.1) Collect the cutting planes from the processors P_i, $2 \leq i \leq N$, until a message indicating end of separation is received from every processor P_i, $2 \leq i \leq N$, and add them to a buffer of inequalities to be added to the LP.

(1.4) If `mode=asynchronous`*, then*

(1.4.1) If the LP solution has been sent in the previous step, or¡if the number of cuts found by standard separation routines is less than a certain number t_2, ($t_1 < t_2$) receive inequalities from the processors $P_i, 2 \leq i \leq N$. Check every inequality for violation and if an inequality is violated by x_k^*, then

- *Add the inequality to a buffer of inequalities to be added to the LP.*
- *If a maximum number of violated inequalities is received, then go to step (1.4.2).*

If a certain time of receiving incoming cutting planes is exceeded, then go to step (1.4.2).

(1.4.2) Send a signal to every processor that no more cutting planes corresponding to the current LP solution should be sent.

(2) Modify the sequential branch-and-cut code in order to send termination signals to all processors $P_i, 2 \leq i \leq N$, if the set of subproblems of the branch-and-cut tree is empty.

Algorithm 4.2 (Parallel cutting plane generation (processor P_i, $2 \leq i \leq N$))

(1) Initialize data structures corresponding to the subset of facets that the processor is responsible for, and initialize a local pool.

(2) Wait until the next message (either termination signal or LP solution) from processor P_1 is received. If the message is the termination signal, then STOP.

(2.1) If `mode=synchronous`*, then:*

(2.1.1) Check inequalities of local pool for violation. If a cutting plane is found:
- *Send it to processor P_1.*
- *If max_{snd} cutting planes are found, go to step (2.1.3).*

(2.1.2) Execute the SIR-separation procedure. If a violated cutting plane is found:
- *Send it to processor P_1.*
- *Insert the violated inequality into the local pool.*
- *If max_{snd} cutting planes are found, then go to step (2.1.3).*

(2.1.3) Send a signal, indicating the end of separation to processor P_1.

(2.1.4) Go to step (2).

(2.2) If `mode=asynchronous`*, then:*

(2.2.1) Choose the most recently received fractional solution x_k^* for separation.

(2.2.2) Set a boolean variable `sendCuttingPlanes=true`*.*

(2.2.3) Generate violated inequalities from local pool. If a cutting plane is found:
- *Send it to processor P_1.*
- *If max_{snd} cutting planes max_{snd} are sent to P_1, then set* `sendCuttingPlanes=false` *and go to step (2.2.4).*

If during the pool separation the signal is received that no more cutting planes corresponding to current LP solution should be sent, then:
- *Set* `sendCuttingPlanes=false`*.*
- *Go to (2.2.4).*

(2.2.4) Execute the SIR-separation procedure. Abort the separation procedure if the next LP solution can be received. If a cutting plane is found, then
- *If a signal is received that no more cutting planes corresponding to the current LP solution x_k^* should be sent, or if max_{snd} cutting planes are sent to P_1, then set* `sendCuttingPlanes=false`*.*
- *If* `sendCuttingPlanes=true`*, then send the inequality to processor P_1.*
- *Insert the inequality into the local pool.*
- *If max_{gen} cutting planes are found, then terminate the separation of x_k^*.*

(2.2.5) Go to step (2).

In synchronous mode, all inequalities which are received by processor P_1 are violated. Moreover, in our application of SIR-separation, the same cutting plane cannot be generated by two different processors, because all processors deal with different classes of facets. Since the separation phase of processor P_1 is not finished until all individual processors $P_i, 2 \leq i \leq N$, have terminated separation, in order to obtain a good load balance, the execution time of the separation procedure should be nearly the same among all individual separators. A single processor $P_i, 2 \leq i \leq N$, terminates the separation procedure in steps (2.1.1) and (2.1.2) if a maximal number max_{snd} of cutting planes is identified and sent to processor P_1.

If asynchronous mode is chosen, then when LP solution x_k^* is to be separated by processor P_1, an inequality can be received in step (1.4.1) of Algorithm 4.1 which has been generated for a previous LP solution $x_{k-i}^*, i \geq 1$. Then it has to be tested if the inequality is still violated. To avoid this test for inequalities which are generated and received in the same separation phase, corresponding to LP solution x_k^*, it is advantageous to send an identification number of the LP solution with an inequality.

In order to avoid the sending of cutting planes when processor 1 has already finished the separation of an LP solution, a signal is sent in step (1.4.2) of Algorithm 4.1.

In step (2.2.4) of Algorithm 4.2 the separation is continued until a maximal number of cutting planes max_{gen} is generated, even if some of the new cutting planes are not sent to processor P_1 (if `sendCuttingPlanes=false`). The reason is that violated inequalities which are inserted into the pool can be very important for pool separation in a following separation phase. This is especially useful if violation of an inequality can be checked easily. Since inequalities of SIR-relaxations are usually sparse, i.e., they have only few non-zero coefficients, the test of violation can be performed fast.

The distinct advantage of asynchronous communication is that all processors P_i, $2 \leq i \leq N$, are busy most of the time and hence more violated cutting planes can be found.

The union of all local pools of inequalities of all processors P_i, $2 \leq i \leq N$, forms a *global pool* which is distributed among the processors P_i, $2 \leq i \leq N$. In our implementation, a local pool is organized as a ring of fixed size, i.e., if the pool is full, then the oldest inequality is removed in order to insert the new inequality. In addition, we use a hash table to make sure that all inequalities in the pool are unique.

5 Separation Strategies

When designing the separation part of a branch-and-cut algorithm, one has choose between different strategies which can have substantial impact on the performance. We want to compare some strategies by computational experiments.

In our computational experiments we have to find answers to the following questions:

- By facet-enumeration algorithm a large number of classes of facets can be generated which is available for SIR-separation. Which and how many classes should be used for separation? We denote this problem as *facet selection problem.*

- The parallel setup allows for the generation of many violated inequalities. Given a large set of cutting planes generated by SIR-separation, which ones should be selected and should be added to the LP? This is the *cutting plane selection problem.*

- How effective are cutting planes generated by SIR-separation for closing the gap between the value of the LP relaxation and the optimal solution value?

- What is the favorable strategy for solving problems to optimality?

Facet selection

It is a natural question which classes of facets are most or least useful in cutting-plane algorithms. In [CR97b] we have discussed various properties of facets of low-dimensional linear ordering polytopes which may have impact on their importance in practical branch-and-cut algorithms. The criteria are:

- The size of a facet. An exact measure of size is the volume of the facet which is hard to determine in general. Correlated to the volume of a facet is its number of *roots*, i.e., feasible 0/1-vectors contained in it. For combinatorial polytopes this number can differ substantially for different facets.

- The strength of a facet. The strength was introduced in [GH96] and is meant to be a measure of how well a relaxation (given by all facets of the equivalence class) approximates a polyhedron in comparison to a weaker relaxation.

- The distance of a facet to a weaker relaxation.

Cutting plane selection

Let a maximization problem be given with objective $c^T x$ and let $f^T x \leq f_0$ be a valid inequality, which is violated by the LP solution x^*. Possible criteria for cutting plane selection are:

S1: Select at random.

S2: Select with priority of maximal *amount of violation* $\delta = f^T x^* - f_0$.

S3: Select with priority of maximal *distance* d between x^* and the hyperplane $f^T x = f_0$, i.e.,

$$d = \frac{f^T x^* - f_0}{||f||} = \frac{\delta}{||f||}.$$

S4: If $f = kc, k > 0$ holds, then $f^T x \leq f_0$ is obviously the "best" inequality that can be added to the linear program, since f_0 is an upper bound of the objective function which is attained for the roots of $f^T x \leq f_0$. Therefore, an idea is to prefer cutting planes $f^Tx \leq f_0$ with f *as parallel as possible as the objective function* c, i.e., which maximize $\frac{c^Tf}{\|f\|}$. Since for the angle ϕ between c and f we have

$$\cos\phi = \frac{c^Tf}{\|c\|\|f\|},$$

we denote this strategy *"angle"* in the following.

S5: Add cutting planes such that the "expected" next LP solution y^* minimizes c^Tx. Since, of course, y^* is not known before reoptimizing, we take $-f$ as approximating the direction of reoptimization. Let $\delta = f^Tx^* - f_0$. Then with $y^* = x^* - \lambda f$ and $f^Ty^* = f_0$ one obtains $\lambda = \frac{\delta}{f^Tf}$ and hence

$$c^Ty^* = c^Tx^* - \frac{\delta c^Tf}{f^Tf}.$$

While strategy S2 is widely used, strategies S5 and S4 were not explored in practical cutting plane generation algorithms so far.

6 Computational Results for the Linear Ordering Problem

The *linear ordering problem (LOP)* is the task to find in a weighted complete digraph $D_n = (V_n, A_n)$ an acyclic tournament of maximum weight. For previous reports on experience with branch-and-cut algorithms for solving it see [GJR84, Rei85, BM97].

The integer programming (IP) formulation is given as

$$\begin{array}{lll} \max & c^Tx & \\ & x(C) \leq 2 & \text{for all directed cycles } C, |C| = 3, \\ & x_{ij} + x_{ji} = 1 & \text{for all } 1 \leq i < j \leq n, \\ & 0 \leq x \leq 1, & \\ & x \text{ integer.} & \end{array}$$

The *linear ordering polytope* P_{LO}^n is defined as the convex hull of the characteristic vectors of all acyclic tournaments in D_n. For $n = 8$, the polytope P_{LO}^n is investigated in [CR97b]. This polytope has (at least) 12,231 σ-classes of facets. All facets are *trivially liftable*, i.e., facet-defining inequalities for $P_{LO}^{n_0}$ define facets for arbitrary $n \geq n_0$. Hence, the known facets of P_{LO}^8 provide a small instance relaxation.

The inequalities $x(C) \leq 2$ for a directed cycle C of length 3 are called *3-dicycle inequalities.* The polytope described by the linear constraints of the IP formulation is called *dicycle relaxation* and is denoted by $P_{LO_C}^n$.

The experimental setup

We want to investigate the practical impact of SIR-separation for solving linear-ordering problems. We implemented a branch-and-cut code for the LOP by using ABACUS, Version 1.3 (beta), and by using parts of the branch-and-cut code of [Rei85]. As LP solver we used CLPEX 4.0. All computations were carried out on Motorola Power PC processors of the Parsytec GC parallel computer at the Interdisciplinary Center for Scientific Computing at the University of Heidelberg.

In all experiments, we add at most 300 cutting planes to the LP in one separation phase. The separation of 3-dicycle inequalities is done by exact enumeration. If not enough violated 3-dicycle inequalities are found, we want to use the facets of P_{LO}^{8} for separation.

The set of problem instances for our experiments was chosen to meet the following requirements.

- The problem instances should be hard enough so that by using only 3-dicycle separation their solution takes many branch-and-cut nodes.
- The structure of different problem instances should be distinguishable and representative.
- In order to compare with previous work, among all problems instances should be an instance which has been solved by an earlier branch-and-cut implementation.

We selected the 9 problem instances `econ77`, `EX1`, `EX2`, `EX4`, `EX5`, `EX6`, `atp66`, `atp76`, `randD`. Among these problems `econ77` is an "easy" instance. `EX1`, `EX2`, `EX4`, `EX5`, `EX6` are random problem instances provided by P. Kobylanski [CK96] with 0/1-objective function, each having 50 nodes. `atp66` and `atp76` are problems with 66 and 76 nodes, respectively, having integral objective function coefficients between 0 and 4. These instances are evaluations of the ATP-matches tennis-players in the year 1995. `randD` is a random linear ordering problem on 46 nodes with coefficients varying between 1 and 100.

Let $c_{opt} = \max\{c^T x \mid x \in P_{LO}^n\}$ and $c_C = \max\{c^T x \mid x \in P_{LO_C}^n\}$. c_C is the objective function value of the LP relaxation at the root node of the branch-and-cut tree if all 3-dicycle inequalities are satisfied. If we add further cutting planes and obtain further LP solutions x^* we are interested in the *gap closure* gc which we define as

$$gc = \frac{c^T x^* - (c_{opt} + 1)}{c_C - (c_{opt} + 1)} \cdot 100.$$

The gap closure is the percentage of the closed gap between dicycle relaxation and optimum objective value plus 1. (Since the objective functions are integral, a subproblem can be fathomed if $c^T x < c_{opt} + 1$.) In the following, n_{sub} (n_{sep}, n_{lp}, n_{cut}) denotes the number of nodes in the branch-and-cut tree (number of SIR-separation phases, number of LP reoptimizations, number of cutting planes found).

We are using the processors of the parallel computer in single user mode. Hence, it is reasonable to consider the elapsed wall-clock time instead of the CPU-time. t_{tot} (t_{lp}) is the total wall-clock time (the wall-clock time which is used by the LP solver). Moreover, $t_{lp}^{\%}$ denotes the percentage of execution time used by the LP solver.

Cutting plane selection in sequential case

	Optimum initialization			No initialization		
	n_{sub}	t_{tot}	$t_{lp}^{\%}$	n_{sub}	t_{tot}	$t_{lp}^{\%}$
`EX1`	2,535	7:15:19	93	2,575	7:26:45	92
`EX2`	3,127	10:26:02	94	3,859	12:55:23	93
`EX4`	37	0:06:57	90	37	0:07:00	89
`EX5`	55	0:10:25	92	199	0:36:38	92
`EX6`	19	0:04:10	88	43	0:07:09	89
`atp66`	13	0:06:49	88	73	0:29:25	93
`atp76`	71	0:49:40	94	341	3:51:42	94
`econ77`	5	0:02:40	33	5	0:02:58	32
`randD`	1,937	4:14:11	92	1,939	4:13:11	92
Sum/Avg.	7,799	23:16:13	84.9	9,071	29:50:11	85.1

Table 1: Best sequential solutions.

First, we solved all instances to optimality by separation of 3-dicycle inequalities only running the sequential version of our code on a single processor. Table 1 shows the corresponding results by using a strategy for selecting cutting planes which is best among different strategies to be discussed below. The problems are solved first by initializing the primal bound with the optimum value, second without this initialization. In both cases the number of subproblems of the branch-and-cut tree, the elapsed time (in hh:mm:ss) and the percentage of the total time which is required by the LP solver is shown. It can be seen that, for almost all problem instances, the total time is dominated by solving the linear programs.

In order to avoid effects caused by primal heuristics for finding feasible solutions, we initialize the primal bound in the following always with the optimal objective function value.

Table 2 compares the different strategies with respect to the elapsed time for solving the problem and the number of LP reoptimizations. In the case of using only 3-dicycle inequalities, strategy S2 and strategy S3 are the same. It can be seen that for all problem instances the selection of cutting planes which maximize $c^T f$ (strategy S4) performs best with respect to both criteria. Most remarkable is how poorly a random strategy can perform which can be seen for instance `atp76`. Hence in the following we use strategy S4 ("angle") as default strategy for adding 3-dicycle inequalities. This strategy also was used for the experiments reported in Table 1.

We now turn to the parallel version of our branch-and-cut algorithm.

	S4 ($c^T f$)		S5 ($\delta c^T f$)		S2 (δ)		S1 (random)	
	n_{lp}	t_{tot}	n_{lp}	t_{tot}	n_{lp}	t_{tot}	n_{lp}	t_{tot}
`EX1`	15,838	7:15:19	16,300	7:34:53	17,291	8:13:24	18,065	8:38:50
`EX2`	20,993	10:26:02	21,834	11:02:58	23,823	12:08:31	24,977	13:07:16
`EX4`	244	0:06:57	252	0:07:29	306	0:10:02	365	0:14:08
`EX5`	346	0:10:25	347	0:10:42	407	0:13:08	445	0:15:31
`EX6`	138	0:04:10	152	0:04:33	214	0:07:12	203	0:08:16
`atp66`	112	0:06:49	114	0:07:06	249	0:17:22	278	0:29:28
`atp76`	591	0:49:40	680	0:59:07	1,331	2:10:30	4,186	14:28:05
`econ77`	30	0:02:40	33	0:02:44	72	0:05:01	63	0:04:45
`randD`	9,711	4:14:11	9,912	4:17:20	10,024	4:15:38	10,097	4:20:23
Sum	48,003	23:16:13	49,624	24:26:52	53,717	27:40:48	58,679	41:46:42

Table 2: Cut selection strategies for dicycle separation.

Comparison of facet properties

In [CR97b] we discussed different measures of quality of linear ordering facets. The criteria we were focusing on were the number of roots of a facet, the trivial and dicycle strength of a facet and the (Euclidean) distance of a facet to the dicycle relaxation.

Now we want to compare the effect of selecting facets relative to these criteria. Since we are interested in the behaviour of different facet classes, at this point a measure of effectiveness should be independent of the computing times of the LP solver. Therefore we think a good criterion of effectiveness is the amount of gap closure relative to the number of cutting planes generation phases at the root node of the branch-and-cut tree.

We chose the following experimental set up. We use the 100 best facets relative to the four criteria. If no more violated 3-dicycle inequalities are found, parallel SIR-separation in synchronous mode is invoked on 7 processors of the parallel machine (so-called SIR-separation phase). If more than 300 cutting planes are generated, the 300 best cutting planes relative to the criterion are selected among them and are added to linear program. We stop if either no more cutting planes are found, or if the maximal time for a problem of 48:00h is exceeded or after 500 cutting plane generation phases. The Grasp-parameters are $\alpha = 0.5$ and $r_{gr} = 75$.

Figure 2 and Figure 3 show the gap closure relative to the number of separation phases for two problem instances. The main result, which holds for all problem instances (including the instances for which no figure is displayed) is that by preferring facets with a large number of roots the gap can be closed most effectively. Comparing the further measures gives not a clear picture. For some instances (e.g, `atp66`, `atp76`) dicycle strength and dicycle distance outperform trivial strength, while for other instances (e.g. `EX4`, `randD`) the opposite effect can be observed.

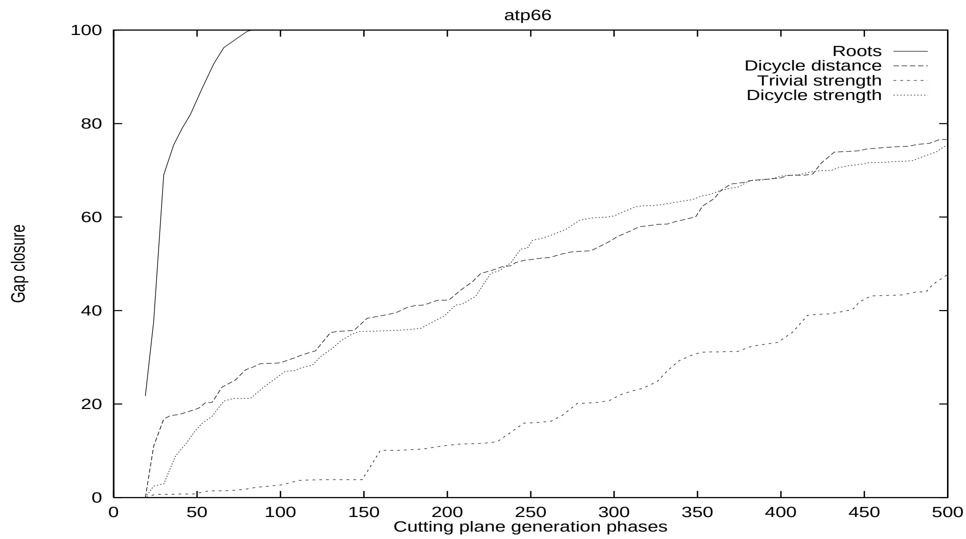

Figure 2: Facet selection relative to various criteria (`atp66`).

Facets with many roots

Since our results show that facets with many roots seem to be good choice to be used for SIR-separation, we now want to compare the "largest" facet classes (with respect to the number of roots) directly.

There are 5 σ-classes of facets having 1008 roots, which is the maximal number of roots of a facet of P_{LO}^8 (except trivial and 3-dicycle facets which both have 20160 roots). The facet classes with 1008 roots are followed by a group of facet classes having 832 roots. Table 3 shows statistics of some facet σ-classes of P_{LO}^8 with many roots. If the facet is defined on 7 or fewer nodes a name (see [Rei93]) is given. The only facet defined on 8 nodes which has 832 roots or more, is the facet with the maximal trivial strength. In some cases the operation of reversing all arcs produces a facet of a different σ-class. If a facet is invariant under that operation, we have a ("Y") in column 4. Moreover, in the table the number of roots, and in addition the trivial strength $s_T(f)$ and dicycle strength $s_C(f)$ and distance to the dicycle relaxation $d_C(F)$ are displayed.

	name	roots	inv.	$s_T(f)$	$s_C(f)$	$d_C(F)$
C1	augmented 3-fence (F_{14})	1008	Y	1.500	1.050	0.129
C2	Möbius ladder (F_4)	1008	N	1.375	1.062	0.151
C3	3-fence (F_3)	1008	N	1.286	1.071	0.167
C4	$\max s_T(f)$	832	Y	1.538	1.038	0.112
C5	(F_{15})	832	N	1.454	1.045	0.125
C6	(F_{16})	832	Y	1.454	1.045	0.125
C7	(F_{12})	832	N	1.400	1.050	0.134
C8	(F_7)	832	Y	1.400	1.050	0.134

Table 3: P_{LO}^8-classes with many roots.

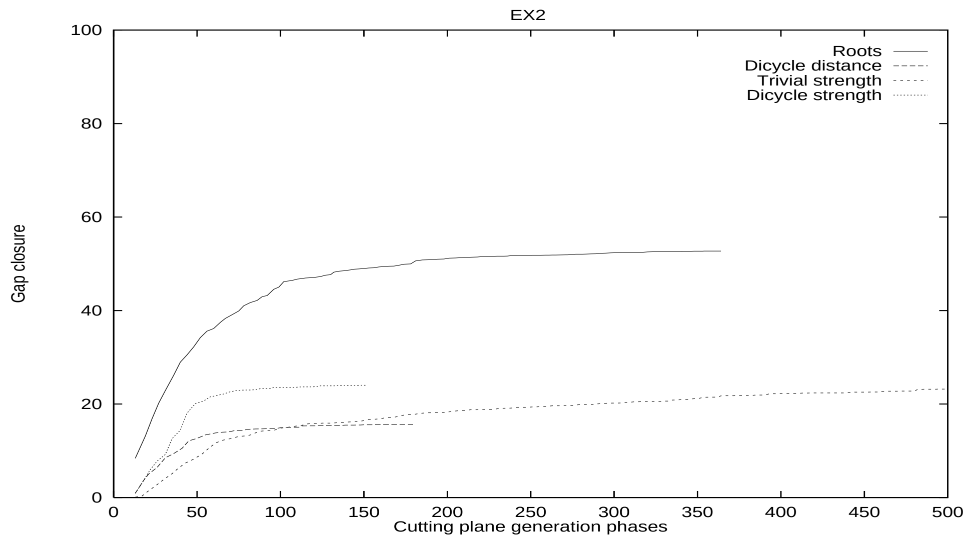

Figure 3: Facet selection relative to various criteria (`EX2`).

We are interested in the gap closure if only one facet class is used for SIR-separation. Figure 4 and Figure 5 show the average gap closure of all problem instances of the test set after the first reoptimization following a SIR-separation phase. (If a problem instance can be solved in fewer than 10 SIR-separation phases then the gap closure is 100, while if we terminate because no more cutting planes are found, we use the last gap closure as value in the following SIR-separation phases.) In order to find a violated inequality with high probability, in this experiment we increased the Grasp parameter r_{gr} to $r_{gr} = 150$.

In Figure 4 it can be seen that with facet class C3 (3-fence inequality) almost no (additional) gap closure can be achieved. The total number of generated violated 3-fence inequalities is only a small fraction (about 4%) of the total number of cutting planes of facet class C1 (or C2). This shows that the number of roots is not a "universal" criterion for a quality of a facet in practical branch-and-cut computations.

The facets with 832 roots perform more regularly, as depicted in Figure 5. In the figure it can be seen that for 2 and more SIR-separation phases, facet classes C5 and C6 which have a trivial strength $s_T(f) = 1.454$ perform better than facet classes C7 and C8 with $s_T(f) = 1.400$.

Consider facet class C4 corresponding to the facet with the maximum trivial strength. While in the first SIR-separation phases only a small gap closure is obtained, for 6 and more SIR-separations this facet class outperforms the other facet classes. A similar behaviour can be observed for class C1, which has the largest trivial strength among the facets with 1008 roots, compared with class C2.

While in previous computations we have shown that facets should be preferred which have a large number of roots, we observe that facets with the same, large number of roots behave different. If many SIR-separation phases are performed, then trivial strength seems to be a criterion of interest.

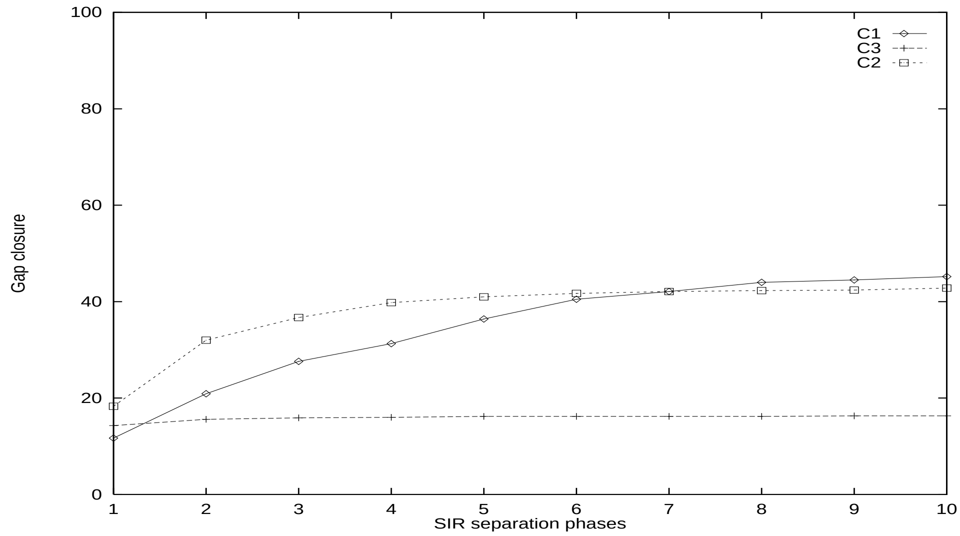

Figure 4: Using single facet classes with 1008 roots.

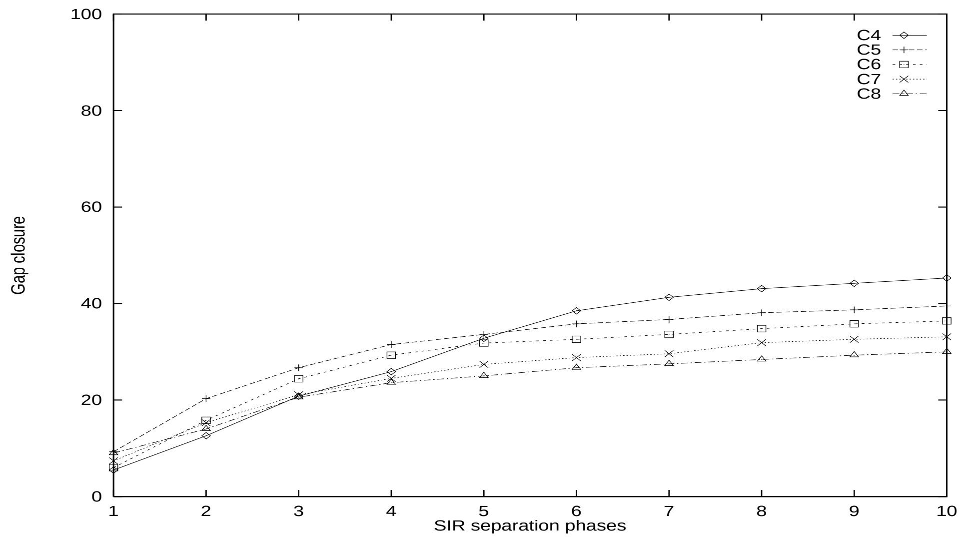

Figure 5: Using single facet classes with 832 roots.

The number of classes for separation

The previous experiments have shown that it is reasonable to prefer σ-classes of facets for separation which have many roots. Now we want to investigate the gap closure at the root node if we vary the number of σ-classes that we use for separation. Now the algorithm terminates if the gap is closed completely or if no more cutting planes are found. If more than 300 cutting planes are found, we select the 300 cutting planes with the maximal violation among them. In the experiments we use alternatively the 7, 15, 100, 1000 best classes of facets with respect to the number of roots.

Figure 6 and 7 display the results for two instances which behave typically. It can be observed that the additional gap closure decreases with increasing separation phases (tailing-off effect). Moreover, it can be seen that when using more facet classes an additional gap closure can be obtained, but the effect when using more facet classes is decreasing.

Our experiments show that the problems `econ77`, `EX6`, `atp66`, `atp76`, `EX4`, `EX5` can be solved at the root node. The main result of these experiments is that, for practical purposes, it seems to be sufficient to use only a small subset of all facet classes for separation.

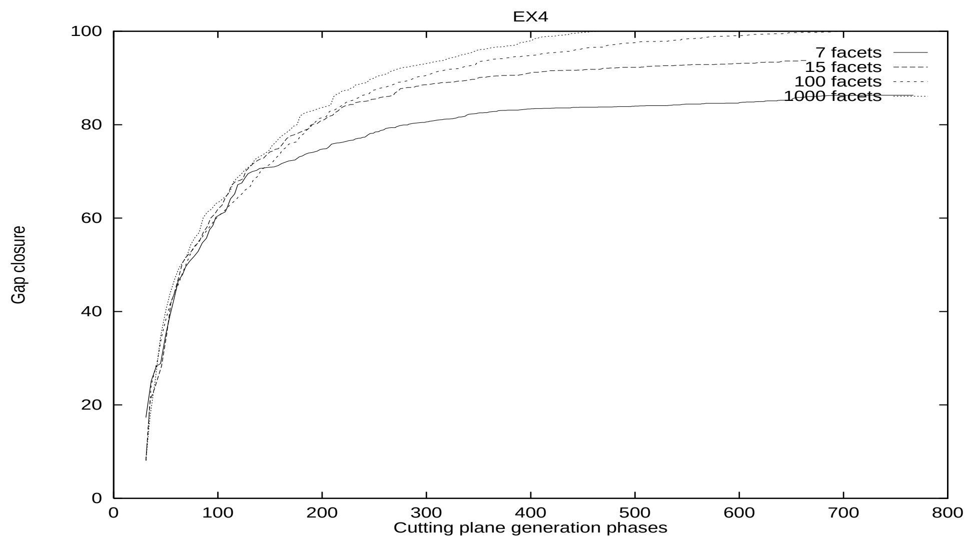

Figure 6: Varying the number of P_{LO}^{8} σ-classes for separation (`EX4`).

Cutting plane selection

As in the case of only separating 3-dicycle inequalities, we want to address the question of which cutting planes should be selected for being added to the linear program among a large set of violated inequalities.

Again we solve all problem instances at the root node. We work with the 14 facet σ-classes which have the largest number of roots among all facet classes. In our experiments we used 7 processors for separation so that every processor had to deal with 2 facets. The parallel SIR-separation is now organized in asynchronous mode. The maximal number of cutting planes to be generated by one processors is $max_{gen} = \lfloor 30000/7 \rfloor$, and the maximal number of cutting planes sent in one separation phase is $max_{snd} = \lfloor 2000/7 \rfloor$. After sending an LP solution, processor P_1 selects the best 300 among all violated inequalities which are received within the next 10 seconds.

Figure 8 displays the average gap closure of all problem instances after the first reoptimization following a SIR-separation phase. We terminate the branch-and-cut algorithm after 20 SIR-separation phases. Observe that selecting cutting planes at random is a bad choice. (This shows that cutting plane selection is a problem necessary to be studied.) For up to 7 SIR-separation phases strategy "angle" is the best choice, followed by strategy "distance", which is slightly better for 8 and more SIR-separation phases. Strategy "roots" is motivated by our previous results that facets with many roots should be preferred. In strategy "roots" the first selection criterion is the number of roots, and among facets with the same number of roots we select with criterion "angle". It can be seen that strategy "roots" is better than a random selection but not as good as strategies "distance" and "angle". It seems that all 14 facet classes which have between 536 and 1008 roots are all in some sense "large" enough, so that it is better to choose among all cutting planes with respect to the "angle" (or "distance").

In principle, the computations confirm the results of separating only 3-dicycle inequalities in the sense that again strategy "angle" is the best choice. However, not for all problem instances strategy "angle" is the best choice, e.g. for instance `randD` we observed that strategy "distance" performs better.

Solving the problems

After investigating different strategies for making use of SIR-separation in an "optimal" way, we now solve the problem instances to optimality.

We use the 14 facet σ-classes with the largest number of roots. Now SIR-separation on 7 processors is invoked if not more than 50 3-dicycle inequalities are violated. The separation is organized in asynchronous mode. The Grasp-parameters are $r_{gr} = 500$, and $\alpha = 0.5$. We use the parameters $max_{gen} = \lfloor 30000/7 \rfloor$ and $max_{snd} = \lfloor 2000/7 \rfloor$ and processor P_1 waits at most 6 seconds for incoming cutting planes.

The best 300 cutting planes with respect to criterion "angle", resp. criterion "distance", are selected in order to add them to the linear program. In strategy "angle" a static rank associated with an inequality is $\frac{c^T f}{||f||}$, and in strategy "distance" we use the distance d between the *last* LP solution and the induced hyperplane $f^T x = f_0$ as its rank. Before pool separation is executed by processors $P_i, i \geq 2$, the inequalities are inserted into a priority queue according to their rank, so that the max_{snd} violated inequalities with the highest rank are sent to processor P_1. At the root node of the branch-and-cut tree two SIR-separation phases are performed and for further subproblems only (at most) one SIR-separation phase is performed. (Computational experiments showed that with more SIR-separations the total number of branch-and-cut nodes can be reduced, but the elapsed total time increases.)

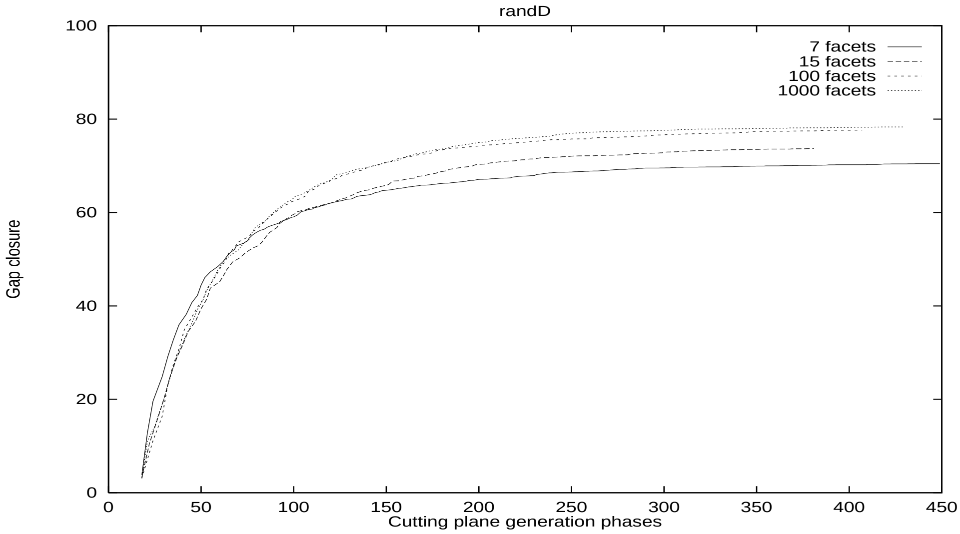

Figure 7: Varying the number of P_{LO}^{8} σ-classes for separation (`randD`).

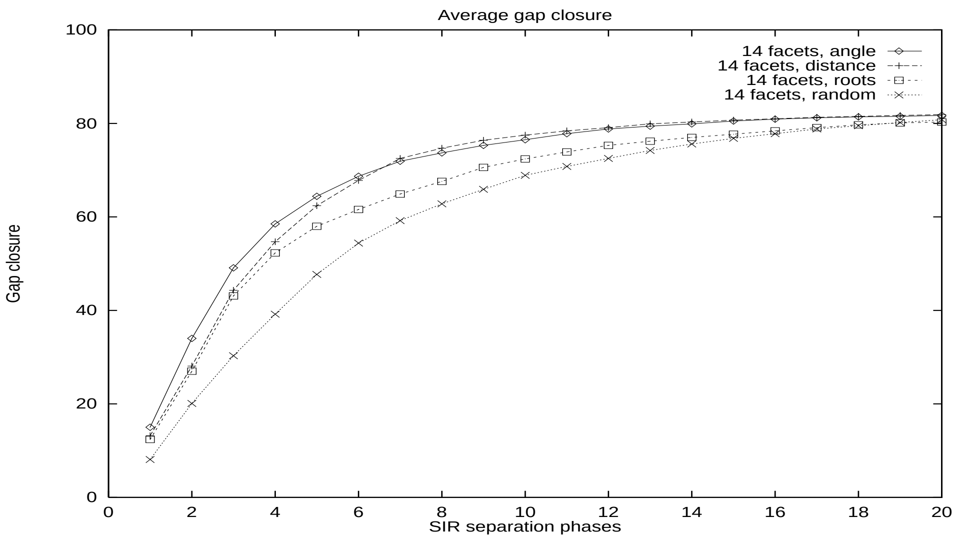

Figure 8: Cutting planes selection strategies.

	t_{tot}	t_{sep}	n_{sub}	n_{lp}	n_{cut}
`EX1`	5:19:17	0:20:31	281	1,425	168,663
`EX2`	6:00:40	0:20:30	261	1,351	161,098
`EX4`	0:09:49	0:00:48	9	63	10,606
`EX5`	0:10:19	0:00:48	11	66	10,182
`EX6`	0:05:46	0:00:27	7	38	6,168
`atp66`	0:10:50	0:00:28	3	39	8,859
`atp76`	1:00:16	0:01:04	11	102	21,411
`econ77`	0:02:35	0:00:25	5	29	5,701
`randD`	1:37:45	0:07:13	79	360	3,501
Sum	14:37:17	0:52:14	667	3,473	427,770

Table 4: Solving the problem instances with cutting plane selection strategy "distance".
14 facet classes, asynchronous mode.

	t_{tot}	t_{sep}	n_{sub}	n_{lp}	n_{cut}
`EX1`	4:17:08	0:17:04	223	1,034	108,426
`EX2`	5:11:16	0:18:22	253	1,158	124,900
`EX4`	0:07:18	0:00:40	9	50	8,061
`EX5`	0:07:45	0:00:46	9	51	7,601
`EX6`	0:04:15	0:00:25	5	31	5,317
`atp66`	0:09:38	0:00:26	3	36	7,918
`atp76`	0:54:16	0:01:13	13	102	20,546
`econ77`	0:02:20	0:00:24	3	25	5,886
`randD`	2:19:05	0:08:54	115	496	42,181
Sum	13:13:01	0:48:14	633	2,983	330,836

Table 5: Solving the problem instances with cutting plane selection strategy "angle".
14 facet classes, asynchronous mode.

Tables 4 and 5 show the results. In addition to the elapsed total time t_{tot}, the elapsed separation time t_{sep} (for processor 1), the number of subproblems n_{sub}, the total number of reoptimizations t_{lp} and the total number of cutting planes added to the linear programs n_{cut} are shown.

All problem instances, except `randD`, can be solved faster with strategy "angle". It can be seen that for all problem instances the separation time is only a small fraction of the total time. Despite SIR-separation, in addition to the separation of 3-dicycle inequalities, the time used by the LP solver is more than 90%.

Compared with the best sequential solutions (Table 2) the number of subproblems of the branch-and-cut tree decreased from 7799 to 633, and in addition the total time could be reduced from 23:16:13h to 13:13:01h. This result is quite satisfactory because frequently one observes in a branch-and-cut algorithm that by adding additional cutting planes (without the strategies we discussed) the number of subproblems can be reduced but the total time increases substantially.

7 Computational Results for the Weighted Betweenness Problem

The weighted betweenness problem arises from an application in computational biology. We refer to [CJK+97] for the biological background and to [COR98] for a more detailed presentation of the IP-model which we consider. We simply define the problem as follows. We are given a set of m so-called *clones* $i \in \{1, 2, \ldots, m\}$ (which are short fragments of a chromosome) to each of which two so-called *end probes* i^t and i^h are associated. These $n = 2m$ probes are numbered such that $i^t = 2i - 1$ and $i^h = 2i$. Now it is of interest to determine certain linear orders of the probes. Namely, depending on some available data, we have for every pair of a clone i and a probe $j \in \{1, 2, \ldots, n\} \setminus \{i^t, i^h\}$ either a betweenness condition (i, j) or a non-betweenness condition $\overline{(i, j)}$. A betweenness condition (i, j) indicates that probe orders are to be preferred where probe j is between the end probes of clone i. Non-betweenness conditions require that some probes should not be between the endprobes of some clones.

The conditions obtained from the input data are usually not consistent, since the data arises from certain experiments which are not free of errors. In particular, there are so-called *false positives* giving rise to betweenness conditions where actually a non-betweenness conditions would be correct and *false negatives* in the opposite case. Therefore a probe order satisfying all conditions usually does not exist. Violation of a betweenness condition is now penalized with cost c_ρ, and violation of a non-betweenness constraint receives a penalty of c_μ. The problem is then to find a linear order of the probes minimizing the sum of penalties for violated constraints. This problem is NP-hard.

The set $\mathcal{B}_m$ of all possible clone-probe pairs is

$$\mathcal{B}_m := \{(i, j) \mid 1 \le i \le m, 1 \le j \le n, j \ne i^t, j \ne i^h\}.$$

With every feasible solution of the problem, corresponding to a permutation π of the probes, we associate a 0/1-vector $\chi^\pi \in \{0, 1\}^{|\mathcal{B}_m|}$

$$\chi^\pi_{ij} = \begin{cases} 1 & \text{if constraint } (i, j) \text{ is met in the order } \pi, \\ 0 & \text{otherwise.} \end{cases}$$

Conversely, there is a fast algorithm that derives from a feasible χ all consistent probe orders.

The betweenness polytope P^m_{BW} is defined as convex hull of all feasible incidence vectors χ. A solution of the Weighted Betweenness Problem can then be obtained finding an optimum vertex solution of

$$\begin{array}{ll} \max & c^T x \\ & x \in P_{BW}^m \\ & x \in \{0,1\}^{2m(m-1)} \end{array}$$

for a certain objective function $c^T x$. Note that since a betweenness condition (i,j) is violated if and only if the non-betweenness condition $\overline{(i,j)}$ is satisfied, one can just complement variables and only speak about betweenness conditions.

We extend our concept of σ-symmetric combinatorial optimization problems. The weighted betweenness problem is a combinatorial optimization problem with ground set $\mathcal{B}_m$. Let $v \in \mathbb{R}^{|\mathcal{B}_m|}$ be an arbitrary vector. If we combine for a pair (i,j) with $1 \leq i,j \leq m$, $i \neq j$, the clone-probe pairs (i,j^t) and (i,j^h) in a 2-dimensional vector, then the entries of v can be placed in an m by m matrix, denoted by $\tilde{v}$, of these 2-dimensional vectors, such that an entry $\tilde{v}_{ij} \in \mathbb{R}^2$ of $\tilde{v}$ is defined as $\tilde{v}_{ij} = \binom{v_{ij^t}}{v_{ij^h}}$, for $i \neq j$. With respect to the set $\mathcal{B}_m$, the rows and columns of the matrix correspond to the clones and an entry ij refers to the relations of the end-probes obtained from clone C_j with clone C_i. We call $\tilde{v}$ the *clone-clone matrix* of v.

For the complete digraph with m nodes, we associate with every edge (i,j) not only one but two elements of the ground set, namely (i,j^t) and (i,j^h). For a vector $v \in \mathbb{R}^{|\mathcal{B}_m|}$, the components v_{i,j^t} and v_{i,j^h} of $\tilde{v}_{ij} \in \mathbb{R}^2$ are associated with an edge (i,j).

For an arbitrary solution of the betweenness problem with corresponding incidence vector χ and for a permutation σ of the m nodes, the vector $\sigma(\chi)$ with $(\widetilde{\sigma(\chi)})_{ij} = \binom{\chi_{\sigma(i)\sigma(j)^t}}{\chi_{\sigma(i)\sigma(j)^h}}$ is also a feasible solution. Therefore, we say that the weighted betweenness problem and the polytope P_{BW}^m are σ-symmetric. This defines the first symmetry property of P_{BW}^m, which corresponds to a simultaneous permutation of the rows and columns of the associated clone-clone matrix $\tilde{\chi}$.

The second symmetry property is that a solution remains feasible if for an arbitrary clone C_j its end-probes are reversed. This corresponds to reversing in column j of the clone-clone matrix the entries of $\binom{\chi_{ij^t}}{\chi_{ij^h}}$ to $\binom{\chi_{ij^h}}{\chi_{ij^t}}$, for all $1 \leq i \leq m, i \neq j$.

We denote the equivalence classes defined by the two symmetry properties P_{BW}^m*-classes*.

P_{BW}^2 has 7 vertices and is completely described by 3 equivalence classes of facets. P_{BW}^3 has 172 vertices and 241 facets in total which can be partitioned into 16 equivalence classes of facets (quoted in [COR98]). The computation of the complete linear description of P_{BW}^4 which has 9197 vertices was not possible in reasonable time. However, by an algorithm for parallel facet enumeration [CR97a] we found more than $1,16 \cdot 10^7$ (!) different equivalence classes of facet-defining inequalities, yielding a lower bound of $4.4 \cdot 10^9$ for the number of facets of P_{BW}^4.

In [COR98] we have shown that P_{BW}^m is full-dimensional and that the facets of P_{BW}^m can be trivially lifted. Hence the facets of P_{BW}^m, $m \leq 4$ which we found provide a small instance relaxation.

SIR-separation

Since P_{BW}^m is a σ-symmetric polytope, and the facet-defining inequalities of P_{BW}^m are trivially liftable, the separation approach of section 3 can be applied. But now, in the quadratic assignment problem every entry is a 2-dimensional vector.

Let an LP solution $x^* \in \mathbb{R}^{|\mathcal{B}_m|}$ and a facet-defining inequality $g^T x \leq g_0$ be given. For a permutation σ of $\{1, \dots, m\}$ we define

$$C(\sigma) = \sum_{i=1}^{m} \sum_{j=1, j\neq i}^{m} \begin{pmatrix} x^*_{ij^t} \\ x^*_{ij^h} \end{pmatrix}^T \begin{pmatrix} g_{\sigma(i)\sigma(j)^t} \\ g_{\sigma(i)\sigma(j)^h} \end{pmatrix}.$$

It is clear that the set $\{\sigma \mid C(\sigma) > g_0\}$ gives all violated inequalities among all inequalities which are (by means of relabeling of the clones) equivalent to $g^T x \leq g_0$. Thus the separation problem for equivalent inequalities (relative to relabeling of the clones) is a quadratic assignment problem in which all entries are 2-dimensional vectors. We tackled this problem by the (modified) Grasp heuristic described before.

Because an inequality $g^T x \leq g_0$ of a small instance relaxation results from trivial node lifting of an inequality $f^T x \leq f_0$ which is facet-defining for $P_{BW}^{\overline{m}}$, $\overline{m} < m$ we have

$$C(\sigma) = \sum_{i=1}^{\overline{m}} \sum_{j=1, j\neq i}^{\overline{m}} \begin{pmatrix} x^*_{\sigma^{-1}(i)\sigma^{-1}(j)^t} \\ x^*_{\sigma^{-1}(i)\sigma^{-1}(j)^h} \end{pmatrix}^T \begin{pmatrix} f_{ij^t} \\ f_{ij^h} \end{pmatrix}.$$

In order to separate all inequalities of one equivalence class this separation procedure has to be executed for at most $2^{\overline{m}}$ different inequalities $f_i^T x \leq f_0^i$ which are equivalent relative to the end probe reversing symmetry; in our computations we randomly generated at most 8 different inequalities in every separation phase.

Further separation procedures

In [COR98] a general class of facet-defining cycle inequalities which can be separated in polynomial time by shortest path computations is described. In addition the separation of infeasible 0/1-vectors is addressed. This is denoted as *integer vector separation* in the following.

Usually, in our experiments these separation procedures are executed only if no cutting planes are found by SIR-separation.

Branch-and-cut-and-price

In the application a clone is a short fragment of the chromosome implying that in a solution of the weighted betweenness problem only a small subset of betweenness variables is one. This suggests to use only a subset of active variables in the linear program and to set the remaining ones to zero. The inactive variables are added dynamically by performing a *pricing step* if necessary.

Namely, given a linear program $\min\{c^T x \mid Ax \leq b, x \geq 0\}$, and a basic LP solution with dual variables y, then the *reduced cost* of a nonbasic variable j is defined as

$$r_j = c_j - y^T A_{.j}.$$

Due to the reduced cost criterion, an inactive variable has to be added if $r_j < 0$.

Let a *clone pair* be a set of variables $\{b_{ij^t}, b_{ij^h}, b_{ji^t}, b_{ji^h}\}$ associated with two clones C_i and C_j. Since in a clone pair either two or none of the variables are one, we add all variables of a clone pair simultaneously to the set of active variables. We initialize the set of active variables with all clone pairs in which at least one variable corresponds to a betweenness constraint and we add a clone pair in a pricing step if at least one of the involved variables has negative reduced cost.

Test problems

For our experiments we generated random problem instances resembling instances occurring in practical applications. To generate the data, the clones were randomly distributed across the chromosome. The *coverage* describes how often a probe along the chromosome is covered by a clone. Betweenness and non-betweenness conditions where then derived from this clone configuration. To create false positives and false negatives, the conditions were then altered with certain probabilites.

A first test set A consists of 150 instance with 42 to 70 probes. Here we used a coverage varying from 3 to 5, a false negative rate of 10%, and a false positive rate varying from 0% to 5%. Across experiments with varying coverage, the clone length was held constant. In the next paragraph, we use the sequential version of our branch-and-cut algorithm for solving the instances of test set A (without using the pricing technique). The computations were carried out on a SUN Sparc IPX station; the times in Tables 6 and 7 are CPU-times.

Afterwards, different strategies for effective parallel SIR-separation are investigated for the instances of a second test set B. This second test set B consists of 8 large random problem instances with coverage 4, 125 clones, a false negative rate of 10%, and a false positive rate of 5%.

Sequential SIR-separation

We always use the classes of facets of P^2_{BW} for separation, since for these facets all possible assignments can be enumerated in time $\mathcal{O}(m^2)$. Only if not enough cutting planes are found by this separation, SIR-separation with facets of P^3_{BW} and P^4_{BW} is invoked.

As for the linear ordering problem, a natural question is which of the facet classes of the polytopes P^2_{BW}, P^3_{BW}, and P^4_{BW} should be used for separation. Among all classes of facets of P^4_{BW} we found 105 additional classes which have more than 400 roots for which we want to compare in detail the results of the heuristic approach to SIR-separation.

In a first experiment we investigated which gap closure can be achieved if only one facet class is used for separation. Now we define the gap closure gc as

$$gc = \frac{c^T x^*}{c_{opt}}.$$

We initialize the primal bound with the optimal objective function value (which we computed in a previous run) and terminate the algorithm at the root node of the branch-and-cut tree if no cutting plane is found. For the linear ordering problem we observed that the number of roots seems to be an important criterion for the importance of a facet-defining inequality in a branch-and-cut algorithm. Therefore Figure 9 shows for all 150 problem instances the average value of gap closure relative to the number of roots. The average value of the gap closure varies between 0.74 and 0.99, but there is no correlation between the number of roots and the gap closure.

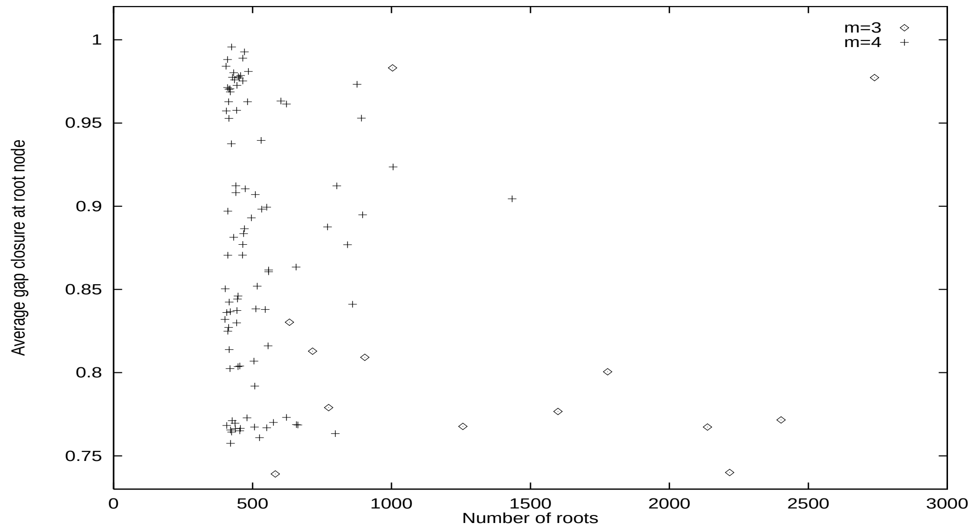

Figure 9: Gap closure at the root node relative to number of roots. A "$\diamond$" corresponds to a facet of P_{BW}^3, and a "+" to a facet defined on 4 nodes.

In order to address the question whether the facet classes with high amount of gap closure should be used with priority for separation, we compare two strategies for selecting the facets. First, we sorted the facet classes. In the first strategy "roots" the criterion for sorting was the number of roots, and in the second strategy "gap closure" it was the observed gap closure. We used the facets of these sorted lists for SIR-separation until at most 300 cutting planes were found. Table 6 shows the average results of solving the 150 problem instances of test set A to optimality. (In that experiment we did not initialize the primal bound with the optimal objective function value, and in addition the 0/1-vector and cycle separation was invoked, if no cuts are found by SIR-separation.)

In Table 6 it can be seen that more cutting planes (n_{cut}) and more reoptimizations (n_{lp}) are needed for the strategy "gap closure" than for strategy "roots". We conclude that if

Strategy	t_{tot}	$t_{lp}^{\%}$	n_{sub}	n_{cut}	n_{lp}
gap closure	0:05:08	5.2%	1.05	1737.8	7.2
roots	0:06:29	2.3%	1.03	1293.8	5.8
special	0:01:28	9.3%	1.13	1022.0	6.3

Table 6: Average values for 150 problems.

cutting planes corresponding to facets with many roots are found, their use seems to be important to close the gap fast.

However, in Table 6 the average total CPU-time t_{tot} is higher for strategy "roots". The reason is that in order to generate a fixed number of cutting planes (we terminate the separation routine if 300 cutting planes are found) more classes of facets have to be considered.

In order to understand why in strategy "roots" more classes of facets are to be considered, in a further experiment, we used every facet class for separation, independently of how many cutting planes of other classes have been generated. Figure 10 displays the average number of cuttting planes relative to the number of roots. It can be seen that for the large facets, which are mainly the facets of P_{BW}^3, not as many cutting planes are found as for facets of P_{BW}^4. An explanation is that by symmetry operations in total more different facets can be generated from one facet representing a facet class of P_{BW}^4 compared with a facet of P_{BW}^3.

We derived a specialized strategy (strategy "special") for sequential SIR-separation by selecting 14 out of 105 facets for using in the separation procedure. The selected facets have many roots and/or are those with many cutting planes in Figure 10 (relative to the other facets). In Table 6 the results are displayed. Since the average number of subproblems increased only slightly, while the total CPU-time could be reduced significantly, the strategy "special" seems to be a good choice for sequential SIR-separation.

SIR-separation compared with conventional separation

Topic of this paper is to propose SIR-separation for solving combinatorial optimization problems with associated σ-symmetric polytopes by branch-and-cut. A question which is to be addressed is: How efficient is SIR-separation compared to other separation procedures? A "conventional" separation procedure would be e.g., the separation of certain cycle inequalities by shortest cycle computation which is described in [COR98].

In computational experiments it turned out that by separating only cycle inequalities the gap cannot be closed sufficiently. If, analogously to the experiments corresponding to Figure 9, in addition to the separation of inequalities of P_{BW}^2 only cycle inequalities are separated, then the average gap closure is only 0.83. Figure 9 shows that for many *single classes* of facets of P_{BW}^m, $m \leq 4$, a higher average gap closure can be achieved.

Nevertheless, cycle separation (and integer vector separation) might be important for finding feasible solutions. Table 7 illustrates this on a typical problem instance. The table shows the results of solving this problem by using different strategies. If for example, only

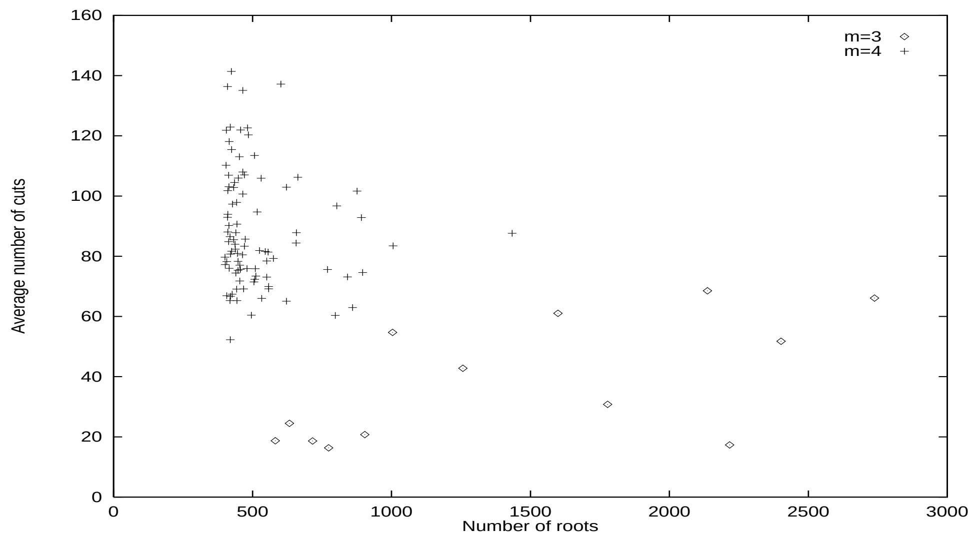

Figure 10: Number of cutting planes relative to number of roots.

SIR-separation with facet ${f^{83}}^T x \le f_0^{83}$ is invoked, (which is the facet with the highest amount of gap closure in Figure 9) the gap could be closed at the root node completely, but in total 55 subproblems have be processed in order to find a feasible solution. Most efficiently the problem could be solved by invoking SIR-separation and in addition cycle separation and integer vector separation. When no SIR-separation was invoked, we gave up after 1,500 subproblems.

The important result is that SIR-separation is essential for closing the gap. However, SIR-separation should be supplemented by using further separation routines.

Separation	n_{cut}				n_{sub}	t_{tot}	gc at root
	$n_{cut}^{P_{BW}^2}$	n_{cut}^{SIR}	n_{cut}^{Cyc}	n_{cut}^{Vtx}			
f^{83}, cycles, integer	17	147	2	2	1	0:00:16	100%
f^{83}, integer	17	147	0	3	3	0:00:18	100%
f^{83}, cycles	19	114	2	0	1	0:00:19	100%
$f^4 - f^{108}$	15	1,378	0	0	1	0:01:04	100%
f^{83}	122	689	0	0	55	0:03:59	100%
cycles, integer	?	?	?	?	$> 1,500$	0:10:00	81%

Table 7: Random problem instance with coverage 3, 21 clones, 0.7% false positives, 10% false negative.
SIR-separation is done with facets ${f^i}^T x \le f_0^i$, "cycles" denotes the separation of cycle inequalities, and "integer" means that infeasible 0/1-vectors are cut off.

Parallel SIR-separation

We now want to discuss our approach to parallel branch-and-cut by solving the problem instances of test set B (eight random problem instances with each having coverage 4, 125 clones, a false negative rate of 10%, and a false positive rate of 5%) with different strategies.

Again, all computations were carried out on the Parsytec GC parallel computer with LP solver CPLEX 4.0 and ABACUS Vers. 1.3 (beta). The computation times are wall-clock times.

First, we solved the problem instances with the sequential version (using one processor of the parallel computer).

Table 8 shows the aggregated results for strategies "roots" and "special". Observe that more than 70% of the elapsed total time (t_{tot}) is needed for SIR-separation (t_{sep}). In addition, in the table the number of facet classes used for separation (n_f), the number of solved LPs (n_{lp}), the total number of added cutting planes (n_{cut}) and the total number of subproblems (n_{sub}) are shown.

	facets	n_f	t_{tot}	t_{sep}	n_{lp}	n_{cut}	n_{sub}
1	roots	105	24:12:18	18:22:04	3,086	837,334	10
2	special	14	20:54:13	14:09:54	3,288	893,774	8

Table 8: Sequential solution of the 8 instances of test set B.
In every cutting plane generation phase separation is stopped if 400 violated inequalities are found.

We solved the problems with 25 different strategies for parallel SIR-separation in asynchronous mode. Table 9 shows the results. The parameter settings I (strategies S1 to S15) aim at spending most time of processor P_1 for solving linear programs while parameter settings II aim at extensive cutting plane generation and selection.

If not more than 100 cutting planes by enumerating facets of P_{BW}^2 are generated (done by processor P_1), the LP solution is sent to processors P_i, $2 \leq i \leq 8$. Then processor P_1 uses the violated inequalities which are received within the next 3 seconds in setting I, and within 5 seconds in setting II. Usually during this time, processors P_i, $2 \leq i \leq 8$ execute pool separation.

In every iteration at most 400 inequalities are added to the linear program. These inequalities are selected from all inequalities received by processor P_1. The maximal number of inequalities sent by all processors $P_i, i \geq 2$, in one separation phase is set to 2000, i.e., $\lfloor 2000/7 \rfloor$ is the maximal number of cutting planes which are sent in every iteration by a single processor. However, in total we allow 5000 in setting I (10000 in setting II) violated inequalities to be generated by processors P_i, $i \geq 2$, all of which are added to the local pool.

Columns 2 to 4 of Table 9 show which and how many facet classes of P_{BW}^4 are used for separation, e.g., in strategy S1, the $n_f = 14$ facets with the largest number of roots are used. "special" denotes the 14 selected classes of facets which we already used in the experiments above, "sparse" denotes the facet classes with the minimal number of non-zeros, and "P_{BW}^3" denotes the facet classes which are defined on three nodes. Every processor has to deal with $\lfloor \frac{n_f}{7} \rfloor$ facets. A "Y" in column 4 means that the $\lfloor \frac{n_f}{7} \rfloor$ facets are selected in random order, i.e., in every SIR-separation phase the next facet class for separation is selected randomly among the $\lfloor \frac{n_f}{7} \rfloor$ facet classes, otherwise the facet classes are processed in order according to their number of roots.

Column 6 shows if variables are generated dynamically ("Y"), or if not ("N"). In the first case, additional pricing steps are performed in every 10th iteration.

Column 7 indicates if the local pools of the processors P_i, $2 \leq i \leq 8$, are sorted according to the cut selection criterion before pool separation is performed.

The elapsed times t_{tot} and t_{sep} (columns 8 and 9) refer to processor 1.

We terminate the branch-and-cut algorithm at the root node; *gc* denotes the average percentage of gap closure.

Computational results with different strategies

Observe that for all but three strategies the gap could be closed completely at the root node. Strategy S9 and S10 show that the cutting planes corresponding to facets of P_{BW}^3 are not sufficient to close the gap completely at the root node; in strategy S4 a time limit of 4:00:00h is exceeded for most of the problem instances.

Comparison of strategies S1, S5, S7, S9 shows that facets with a large number of roots (S1) are the best choice. Note that for facet selection "special" (S5) less reoptimizations (n_{lp}) and less cutting planes (n_{cut}) are needed, but the total time is considerably higher. Strategies S11, S12, S13 confirm the result that facet selection "roots" is the best choice, compared with strategies "special" and "sparse".

Analogously to the experiments for the linear ordering problem we compare different strategies for cutting plane selection (see column 5). In strategy "dist" we want to maximize the distance of the LP solution to the supporting hyperplane, "angle" prefers cutting planes which are parallel to the objective function, i.e., with a high value of $-\frac{c^T f}{||f||}$ (where $c^T x$ is the objective function to be minimized and $f^T x \leq f_0$ is the violated inequality). In addition, "roots" denotes a strategy in which the number of roots of a facet is the first criterion, and the amount of $-\frac{c^T f}{||f||}$ is the second criterion and "rand" denotes a random selection.

By comparing strategies S1 to S4 it is evident that strategy "angle" is the best choice.

Observe that $-\frac{c^T f}{||f||}$ is a value which is independent of the LP solution. This allows us to use this value as a *rank* for every inequality. When pool separation is executed the inequalities can be sorted according to this rank (indicated by a "Y" in column 7). The result is that the violated inequalities of the pool with the highest rank are found. The effect is considerable (compare strategies S1 and S11, or S5 and S12, or S7 and S13). We observed that almost all cutting planes which are sent to P_1 are identified by pool separation.

Now compare strategies S11 and S16. It can be seen that extensive cutting plane generation (S16) does pay off, since the total time can be reduced, even if much more time is spent for separation.

	facet selection			cut							
	facets	n_f	rd	sel.	pr	SP	t_{tot}	t_{sep}	n_{lp}	n_{cut}	gc
					parameter settings I						
S1	roots	14	N	angle	Y	N	4:34:39	0:46:41	1,019	298,965	100.0
S2	roots	14	N	roots	Y	N	5:57:35	1:37:20	1,726	586,894	100.0
S3	roots	14	N	dist	Y	N	10:55:36	1:22:14	1,613	553,640	100.0
S4	roots	14	N	rand	Y	N	29:55:03	1:37:59	1,951	676,156	99.6
S5	special	14	N	angle	Y	N	7:02:01	0:45:16	1,002	294,158	100.0
S6	special	14	N	dist	Y	N	7:03:19	0:53:08	1,073	348,635	100.0
S7	sparse	14	N	angle	Y	N	5:44:43	0:46:09	1,053	305,510	100.0
S8	sparse	14	N	dist	Y	N	8:13:23	1:09:13	1,369	450,855	100.0
S9	P^3_{BW}	13	N	angle	Y	N	9:28:11	1:30:02	1,737	582,995	99.8
S10	P^3_{BW}	13	N	dist	Y	N	8:56:31	1:42:37	1,969	660,744	99.7
S11	roots	14	N	angle	Y	Y	2:06:41	0:41:50	877	249,935	100.0
S12	special	14	N	angle	Y	Y	2:34:30	0:39:48	878	253,491	100.0
S13	sparse	14	N	angle	Y	Y	2:31:42	0:43:52	927	261,580	100.0
S14	roots	14	N	roots	Y	Y	2:14:06	0:54:52	1,040	333,601	100.0
S15	roots	14	N	angle	N	Y	2:13:25	0:44:51	763	244,024	100.0
					parameter settings II						
S16	roots	14	N	angle	Y	Y	2:02:11	1:02:60	808	231,811	100.0
S17	roots	21	N	angle	Y	Y	2:02:22	1:03:12	815	235,134	100.0
S18	roots	28	N	angle	Y	Y	2:01:34	1:02:01	802	237,181	100.0
S19	roots	105	N	angle	Y	Y	2:06:09	1:06:04	836	231,779	100.0
S20	roots	14	N	roots	Y	Y	2:43:23	1:25:28	1,042	321,347	100.0
S21	roots	21	N	roots	Y	Y	2:58:46	1:33:43	1,133	348,743	100.0
S22	roots	14	Y	angle	Y	Y	2:21:60	0:54:39	778	211,969	100.0
S23	roots	21	Y	angle	Y	Y	2:38:27	0:57:43	837	232,630	100.0
S24	roots	28	Y	angle	Y	Y	3:26:46	1:06:03	956	274,554	100.0
S25	roots	28	N	angle	N	Y	2:09:14	1:04:43	712	217,933	100.0

Table 9: Parallel solution of test set B with 8 processors.

It should be noted that both criteria, the number of roots of a facet and its angle with the objective function, are important. Strategy S20 and S21 show that the number of roots of a facet alone is not a good choice, and strategies S22 to S24 show that it is not a good choice either to select facets for separation which are not the largest ones.

To see the effect of adding variables dynamically, strategy S15 is a variation of strategy S11, and strategy S25 is a variation of the "optimal" strategy S18. While the problem instances of test set B can be solved with the complete variables set, with the pricing technique the solutions are obtained slighly faster. In the case of using the pricing technique the linear programs are initialized with about 25% of all variables, and about 5% of all variables are added dynamically.

By parallel branch and cut on 8 processors the total computation time for solving the test set B could be reduced from 20:54:13h for the sequential algorithm to 2:01:34h. This superlinear speed up results from using large cutting plane pools and sophisticated cutting plane generation and selection strategies. The best strategy is to select cutting planes $f^T x \leq f_0$ which are large (with respect to the number of roots) and whose normal vector points in the same direction as the objective function c, i.e., which have a large value of $-\frac{c^T f}{||f||}$. Compared with the sequential solutions (without cutting plane selection), the problem instances could all be solved to optimality at the root node and the total number of reoptimizations and cutting planes added to the linear program is decreased by about 75%.

8 Conclusion

In this paper we have shown that small instance relaxations proved to be very useful for the practical solution of hard combinatorial optimization problems with a branch-and-cut-approach. For the linear ordering and the weighted betweenness problem, solution times could be substantially improved compared to standard branch-and-cut approaches.

Determining small instance relaxations usually yields a host of facet classes that are in principle suited for being incorporated into a branch-and-cut algorithm. Since, in view of running time, it is not reasonable to take all these classes into account, we performed extensive experiments to address in detail the facet selection problem (which facet classes should be used for separation) and the cutting plane selection problem (which subset of all identified violated inequalities should be used). Concerning the first problem, it turned out that facet classes with a large number of roots are the most important ones. For the second problem it is advisable to give priority to cutting planes whose normal vector is almost parallel to the objective function gradient.

A particular feature of small instance relaxations is that the separation problem for the various facet classes can be attacked in a general way by considering quadratic assignment problems. Moreover, separation can be parallelized with a straightforward approach and parallel hardware can be used effectively.

We believe that the use of small instance relaxations can substitute a lot of theoretical investigations that would otherwise have to be conducted when designing a branch-and-cut algorithm for a hard (symmetric) combinatorial optimization problem. Based on our experiments and since powerful software for computing linear descriptions of polytopes is available, time for implementing first solution algorithms can be substantially reduced. And, even already implemented branch-and-cut codes can benefit from the additional incorporation of separation based on small instance relaxations.

References

[BM97] B. Borchers and J.E. Mitchell. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. Technical report, Rensselaer Polytechnic Institute, 1997.

[CJK+97] T. Christof, M. Jünger, J. Kececioglu, P. Mutzel, and G. Reinelt. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. *Journal of Computational Biology*, 4(4):433–447, 1997.

[CK96] S. Chanas and P. Kobylanski. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6:191–205, 1996.

[COR98] T. Christof, M. Oswald, and G. Reinelt. Consecutive ones and a betweenness problem in computational biology. In *Integer Programming and Combinatorial Optimization*, 1998. Proceedings of the 6th International IPCO Conference.

[CR95] T. Christof and G. Reinelt. Parallel cutting plane generation for the TSP. In P. Fritzson and L. Finmo, editors, *Parallel Programming and Applications*. IOS Press, 1995.

[CR96] T. Christof and G. Reinelt. Combinatorial optimization and small polytopes. *Top*, 4(1):1–64, 1996.

[CR97a] T. Christof and G. Reinelt. Efficient parallel facet enumeration for 0/1 polytopes. Technical report, University of Heidelberg, Germany, 1997.

[CR97b] T. Christof and G. Reinelt. Low-dimensional linear ordering polytopes. Technical report, University of Heidelberg, Germany, 1997.

[GH96] M.X. Goemans and L.A. Hall. The strongest facets of the acyclic subgraph polytope are unknown. In W.H. Cunningham, S.T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science 1084, pages 415–429. Springer, 1996.

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[JRT95] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In W. Cook, L. Lovász, and P. Seymour, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 20: Combinatorial Optimization*. American Mathematical Society, 1995.

[JT97a] M. Jünger and S. Thienel. The design of the branch-and-cut system ABACUS. Technical Report 95.260, Universität zu Köln, Germany, 1997.

[JT97b] M. Jünger and S. Thienel. Introduction to ABACUS - A Branch-And-CUt system. Technical Report 95.263, Universität zu Köln, Germany, 1997.

[Kai97] V. Kaibel. *Polyhedral Combinatorics of the Quadratic Assignment Problem.* PhD thesis, Universität zu Köln, 1997.

[LPR94] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 16*, pages 237–261. American Mathematical Society, 1994.

[NSS94] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

[PR96] M. Padberg and M. Rijal. *Location, Scheduling, Design and Integer Programming.* Kluwer Academic Publishers, 1996.

[PRW94] P.M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P.M. Pardalos and H. Wolkowicz, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 16.* American Mathematical Society, 1994.

[Rei85] G. Reinelt. *The Linear Ordering Problem: Algorithms and Applications.* research and exposition in mathematics 8. Berlin: Heldermann, 1985.

[Rei93] G. Reinelt. A note on small linear ordering polytopes. *Discrete & Computational Geometry*, 10:67–78, 1993.

[Thi95] S. Thienel. *ABACUS A Branch-And-CUt System.* PhD thesis, Universität zu Köln, 1995.