BRANCH-AND-CUT ALGORITHMS FOR INTEGER PROGRAMMING, *Branch-and-cut*

Branch-and-cut methods are exact algorithms for *integer programming problems*. They consist of a combination of a **cutting plane method** with a **branch-and-bound algorithm**. These methods work by solving a sequence of linear programming relaxations of the integer programming problem. Cutting plane methods improve the relaxation of the problem to more closely approximate the integer programming problem, and branch-and-bound algorithms proceed by a sophisticated divide and conquer approach to solve problems. The material in this entry builds on the material contained in the entries on cutting plane and branch-and-bound methods.

Perhaps the best known branch-and-cut algorithms are those that have been used to solve *traveling salesman problems*. This approach is able to solve and prove optimality of far larger instances than other methods. Two papers that describe some of this research and also serve as good introductions to the area of branch-and-cut algorithms are [21, 32]. A more recent work on the branch-and-cut approach to the traveling salesman problem is [1]. Branch-and-cut methods have also been used to solve other combinatorial optimization problems; recent references include [8, 10, 13, 23, 24, 26]. For these problems, the cutting planes are typically derived from studies of the polyhedral combinatorics of the corresponding integer program. This enables the addition of strong cutting planes (usually facet defining inequalities), which make it possible to considerably reduce the size of the branch-and-bound tree. Far more detail about these strong cutting planes can be found elsewhere in this encyclopedia, for example in the entry on **cutting plane methods for integer programming**.

integer programming problems
cutting plane method
branch-and-bound algorithm
traveling salesman problems
cutting plane methods for integer programming
linear programming relaxation
LP relaxation

Branch-and-cut methods for general integer programming problems are also of great interest (see, for example, the papers [4, 7, 11, 16, 17, 22, 28, 30]). It is usually not possible to efficiently solve a general integer programming problem using just a cutting plane approach, and it is therefore necessary to also to branch, resulting in a branch-and-cut approach. A pure branch-and-bound approach can be sped up considerably by the employment of a cutting plane scheme, either just at the top of the tree, or at every node of the tree.

For general problems, the specialized facets used when solving a specific combinatorial optimization problem are not available. Some useful families of general inequalities have been developed; these include cuts based on knapsack problems [17, 22, 23], Gomory cutting planes [19, 20, 5, 12], lift and project cutting planes [29, 33, 3, 4], and Fenchel cutting planes [9]. All of these families of cutting planes are discussed in more detail later in this entry.

The software packages MINTO [30] and ABACUS [28] implement branch-and-cut algorithms to solve integer programming problems. The packages use standard linear programming solvers to solve the relaxations and they have a default implementation available. They also offer the user many options, including how to add cutting planes and how to branch.

A simple example.

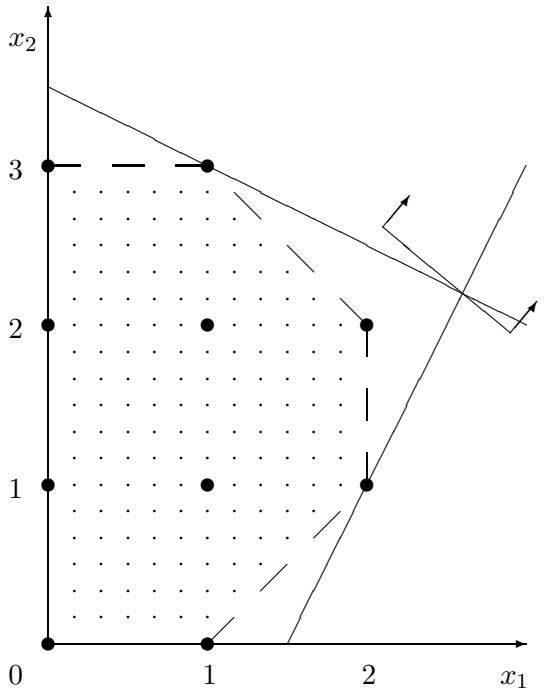
Consider the integer programming problem

$$\begin{array}{lrcrcl} \min & -5x_1 & - & 6x_2 & & \\ \text{s.t.} & x_1 & + & 2x_2 & \leq & 7 \\ & 2x_1 & - & x_2 & \leq & 3 \\ & & & x_1, x_2 & \geq & 0, \text{ integer.} \end{array}$$

This problem is illustrated in the figure. The feasible integer points are indicated. The *linear programming relaxation* (or *LP relaxation*)

is obtained by ignoring the integrality restrictions; this is given by the polyhedron contained in the solid lines.

The first step in a branch-and-cut approach is to solve the linear programming relaxation, which gives the point $(2.6, 2.2)$, with value -26.2. There is now a choice: should the LP relaxation be improved by adding a cutting plane, for example, $x_1 + x_2 \leq 4$, or should the problem be divided into two by splitting on a variable?



A branch-and-cut example

Assume the algorithm makes the second choice, and further assume that the decision is to split on x_2, giving two new problems:

$$\begin{array}{llrcl} \min & -5x_1 & - \quad 6x_2 & & \\ \text{s.t.} & x_1 & + \quad 2x_2 & \leq & 7 \\ & 2x_1 & - \quad x_2 & \leq & 3 \\ & & \mathbf{x_2} & \geq & \mathbf{3} \\ & & x_1, x_2 & \geq & 0, \text{ integer} \end{array}$$

and

$$\begin{array}{llrcl} \min & -5x_1 & - \quad 6x_2 & & \\ \text{s.t.} & x_1 & + \quad 2x_2 & \leq & 7 \\ & 2x_1 & - \quad x_2 & \leq & 3 \\ & & \mathbf{x_2} & \leq & \mathbf{2} \\ & & x_1, x_2 & \geq & 0, \text{ integer.} \end{array}$$

The optimal solution to the original problem will be the better of the solutions to these two subproblems. The solution to the linear programming relaxation of the first subproblem is $(1, 3)$, with value -23. Since this solution is integral, it solves the first subproblem. This solution becomes the incumbent best known feasible solution. The optimal solution for the linear programming relaxation of the second subproblem is $(2.5, 2)$, with value -24.5. Since this point is nonintegral, it does not solve the subproblem. Therefore, the second subproblem must be attacked further.

It is possible to branch using x_1 in the second subproblem; instead, assume the algorithm uses a cutting plane approach and adds the inequality $x_1 + 2x_2 \leq 6$. This is a valid inequality, in that it is satisfied by every integral point that is feasible in the second subproblem. Further, this inequality is a cutting plane: it is violated by $(2.5, 2)$. Adding this inequality to the relaxation and resolving gives the optimal solution $(2.4, 1.8)$, with value -22.6. The subproblem still does not have an integral solution. However, notice that the optimal value for this modified relaxation is larger than the value of the incumbent solution. The value of the optimal integral solution to the second subproblem must be at least as large. Therefore, the incumbent solution is better than any feasible integral solution for the second subproblem, so it actually solves the original problem.

Of course, there are several issues to be resolved with this algorithm, including the major questions of deciding whether to branch or to cut and deciding how to branch and how to generate cutting planes. Notice that the cutting plane introduced in the second subproblem is not valid for the first subproblem. This inequality can be modified to make it valid for the first subproblem by using a *lifting* technique, which is discussed later in this entry.

A standard form.

To fix notation, the following problem is regarded as the standard form mixed integer linear

programming problem in this entry:

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & \leq & b \\ & x & \geq & 0 \\ & x_i & & \text{integer, } i = 1, \ldots, p. \end{array}$$

Here, x and c are n-vectors, b is an m-vector, and A is an $m \times n$ matrix. The first p variables are restricted to be integer, and the remainder may be fractional. If $p = n$ then this is an integer programming problem. If a variable is restricted to take the values 0 or 1 then it is a binary variable. If all variables are binary then the problem is a binary program.

Primal heuristics.

In the example problem, it was possible to prune the second subproblem by bounds, once an appropriate cutting plane had been found. The existence of a good incumbent solution made it possible to prune in this way. In this case, the solution to the linear programming relaxation of the first subproblem was integral, providing the good incumbent solution. In many cases, it takes many stages until the solution to a relaxation is integral. Therefore, it is often useful to have good heuristics for converting the fractional solution to a relaxation into a good integral solution that can be used to prune other subproblems. Primal heuristics are discussed further in the entry on **branch-and-bound algorithms**.

Preprocessing.

A very important component of a practical branch-and-cut algorithm is preprocessing to eliminate unnecessary constraints, determine any fixed variables, and simplify the problem in other ways. Preprocessing techniques are discussed in the entry on **branch-and-bound algorithms**.

Families of cutting planes.

Perhaps the first family of cutting planes for general mixed integer programming problems were *Chvátal-Gomory cutting planes* [19, 20, 15]. These inequalities can be derived from the final tableau of the linear programming relaxation, and they are discussed in more detail in the entry on **cutting plane algorithms**. These cuts can be useful if they are applied in a computationally efficient manner [5, 12]. Gomory cuts can contain a large number of nonzeroes, so care is required to ensure that the LP relaxation does not become very hard with large memory requirements. The cuts are generated directly from the basis inverse, so care must also be taken to avoid numerical difficulties.

One of the breakthroughs in the development of branch-and-cut algorithms was the paper by `H. P. Crowder`, `E. L. Johnson`, and `M. Padberg` [17]. This paper showed that it was possible to solve far larger general problems than had previously been thought possible. The algorithm used extensive preprocessing, good primal heuristics, and cutting planes derived from *knapsack problems* with binary variables. Any inequality in binary variables can be represented as a knapsack inequality $\sum_{i \in N} a_i x_i \leq b$ with all $a_i > 0$ for some subset N of the variables, eliminating variables or replacing a variable x_j by $1 - x_j$ as necessary. The facial structure of the knapsack polytope can then be used to derive valid inequalities for the problem. For example, if $R \subseteq N$ with $\sum_{i \in R} a_i > b$ then $\sum_{i \in R} x_i \leq |R| - 1$ is a valid inequality. Further, if R is a minimal such set, so deleting any member of R leaves the sum of coefficients smaller than b, then the inequality defines a facet of the corresponding knapsack polytope. Other inequalities can be derived from the knapsack polytope. These inequalities have been extended to knapsacks with general integer variables and one continuous variable [11] and to binary problems with generalized upper bounds [34].

branch-and-bound algorithms
branch-and-bound algorithms
Chvátal-Gomory cutting planes
cutting plane algorithms
`H. P. Crowder`
`E. L. Johnson`
`M. Padberg`
knapsack problems

Another family of useful inequalities are *lift-and-project* or *disjunctive inequalities*. These were originally introduced by E. Balas [2], and it is only in the last few years that the value of these cuts has become apparent for general integer programming problems [3, 4]. Given the feasible region for a binary programming problem $S := \{x : Ax \leq b, x_i = 0, 1\ \forall i\}$, each variable can be used to generate a set of disjunctive inequalities. Let $S_j^0 := \{x : Ax \leq b, 0 \leq x_i \leq 1\ \forall i, x_j = 0\}$ and $S_j^1 := \{x : Ax \leq b, 0 \leq x_i \leq 1\ \forall i, x_j = 1\}$. Then $S \subseteq S_j^0 \cup S_j^1$, so valid inequalities for S can be generated by finding valid inequalities for the convex hull of $S_j^0 \cup S_j^1$. These inequalities are generated by solving linear programming problems. Because of the expense, the cuts are usually only generated at the root node. Nonetheless, they can be very effective computationally.

Other general cutting planes have been developed. The paper [16] describes several families and discusses routines for identifying violated inequalities. Fenchel cutting planes, which are generated using ideas from Lagrangian duality and convex duality, are introduced in [9].

When to add cutting planes.

The computational overhead of searching for cutting planes can be prohibitive. Therefore, it is common to not search at every node of the tree. Alternatives include searching at every eighth node, say, or at every node at a depth of a multiple of eight in the tree.

Generally, at each node of the branch-and-bound tree, the linear programming relaxation is solved, cutting planes are found, these are added to the relaxation, and the process is repeated. Usually, there comes a point at which the process tails off, that is, the solution to one relaxation is not much better than the solutions to the recent relaxations. It is then advisable to stop work on this node and branch. Tailing off is a function of lack of knowledge about the polyhedral structure of the relaxation, rather than a fundamental weakness of the cutting plane approach [32]. In some implementations, a fixed number of rounds of cutting plane searching are performed at a node, with perhaps several rounds performed at the root node, and fewer rounds performed lower in the tree.

The *cut-and-branch* variant adds cutting planes only at the root node of the tree. Usually, an implementation of such a method will expend a great deal of effort on generating cutting planes, requiring time far greater than just solving the relaxation at the root. The benefits of cut-and-branch include

- all generated cuts are valid throughout the tree, since they are valid at the root.
- bookkeeping is reduced, since the relaxations are identical at each node.
- no time is spent generating cutting planes at other nodes.

Cut-and-branch is an excellent technique for many general integer programs, but it lacks the power of branch-and-cut for some hard problems. See [16] for more discussion of the relative computational performance of cut-and-branch and branch-and-cut.

Lifting cuts.

A cut added at one node of the branch-and-cut tree may well not be valid for another subproblem. Of course, it is not necessary to add the cut at any other node, in which case the cut is called a *local* cut. This cut will then only affect the current subproblem and its descendants. The drawback to such an approach is in the potential memory requirement of needing to store a different version of the problem for each node of the tree. In order to make a cut valid throughout the tree (or *global*), it is necessary to *lift* it.

Lifting can be regarded as a method of rotating a constraint. Returning to the example problem once again, the constraint $x_1 + 2x_2 \leq 6$ is valid if $x_2 \leq 2$. To extend this constraint to

lift-and-project
disjunctive inequalities
E. Balas
cut-and-branch
global→ global cuts
lift→ lifting cuts

be valid when $x_2 \geq 3$, consider the inequality

$$x_1 + 2x_2 + \alpha(x_2 - 2) \leq 6.$$

It is desired to take α as large as possible while ensuring that this is a valid inequality. If $x_2 = 3$ then $x_1 \leq 1$, so the inequality is valid for $x_2 = 3$ provided $\alpha \leq -1$. If $x_2 = 1$, the inequality is valid provided $\alpha \geq -2$. Finally, the inequality is valid when $x_2 = 0$ provided $\alpha \geq -2.5$. Combining these conditions gives that the valid range is $-2 \leq \alpha \leq -1$. The two extreme choices $\alpha = -1$ and $\alpha = -2$ give the valid inequalities $x_1 + x_2 \leq 4$ and $x_1 \leq 2$, respectively. Other valid choices for α give inequalities that are convex combinations of these two. In this way, valid inequalities for one node of the tree can be extended to valid inequalities at any node.

See [11] for more discussion of lifting in the case of general mixed integer linear programming problems. It is often not possible to lift inequalities for such problems because the upper and lower bounds on the coefficients conflict. Of course, if an inequality is valid at the root node then it is valid throughout the tree so there is no need to lift. This is one of the reasons why general inequalities such as Chvátal-Gomory cuts or lift-and-project cuts are often more successfully employed in a cut-and-branch approach.

The method of calculating coefficients in the case of binary problems is now outlined — see [31] for more details. The inequality generated at a node in the tree will generally only use the variables that are not fixed at that node. Lifting can be used to make the inequality valid at any node of the tree. It is necessary to apply the lifting process for each variable that has been fixed at the node, examining the opposite value for that variable. For example, if the inequality

$$\sum_{j \in J} a_j x_j \leq h \text{ for some subset } J \subseteq \{1, \ldots, n\}$$

is valid at a node where x_i has been fixed to zero, the lifted inequality takes the form

$$\sum_{j \in J} a_j x_j + \alpha_i x_i \leq h$$

for some scalar α_i. This scalar should be maximized in order to make the inequality as strong as possible. Now, maximizing α_i requires solving another integer program, so it may be necessary to make an approximation. This process has to be applied successively to each variable that has been fixed at the node. The order in which the variables are examined may well affect the final inequality, and other valid inequalities can be obtained by lifting more than one variable at a time.

Implementation details.

Many details of tree management can be found in the entry on **branch-and-bound algorithms**. This includes node selection, branching variable selection, and storage requirements, among other issues. Typically, a branch-and-bound algorithm stores the solution to a node as a list of the indices of the basic variables. Branch-and-cut algorithms may require more storage if cuts are added locally, because it is then necessary to be able to recreate the current relaxation at any active node with just the appropriate constraints. If cuts are added globally, then it suffices to store a single representation of the problem.

It is possible to fix variables using information about reduced costs and the value of the best known feasible integral solution, as described in the entry on **cutting plane algorithms**. Once variables have been fixed in this way, it is often possible to fix additional variables using logical implications. In order to fully exploit the fixing of variables, *parent node reconstruction* [32] is performed as follows. Once a parent node has been selected, it is not immediately divided into two children, but is solved again using the cutting plane algorithm. When the cutting plane procedure terminates, the optimal reduced cost vector has been reconstructed and this is used to perform variable fixing.

Many branch-and-cut implementations use a *pool of cuts* [32]. This is typically a set of constraints that have been generated earlier and

branch-and-bound algorithms
cutting plane algorithms

either not included in the relaxation or subsequently dropped because they no longer appeared to be active. It is easy to check these cuts for violation and this is usually done before more involved separation routines are invoked. The pool of cuts also makes it possible to reconstruct the parent node more efficiently, partly because difficulties with tailing off are reduced.

Solving large problems.

The difficulty of a particular integer programming problem is not purely a function of the size of the problem. There are problems in the MIPLIB test set [6] with just a few hundred variables that prove resistant to standard solution approaches. The difficulty is caused by an explosion in the size of the tree.

For some problems, difficulties are caused by the size of the LP relaxation, and *interior point methods* may be useful in such cases. Interior point methods are superior to simplex methods for many linear programming problems with thousands of variables. However, restarting is harder with an interior point method than with a simplex method when the relaxation is only slightly changed. Therefore, for very large problems, the first relaxation at the top node of the tree can be solved using an interior point method, and subsequent relaxations can be solved using the (dual) simplex method. For some problems, the relaxations are just too large to be handled with a simplex method. For example, the relaxations of the *quadratic assignment problem* given in [25] were solved using interior point methods. Interior point methods also handle degeneracy better than the simplex method. Therefore, the branch-and-cut solver described in [1] occasionally uses an interior point method to handle some subproblems.

One way to enable the solution of far larger problems is to use a *parallel computer*. The nature of branch-and-cut and branch-and-bound algorithms makes it possible for them to exploit coarse grain parallel computers efficiently: typically, a linear programming relaxation is solved on a node of the computer. It is possible to use one node to manage the distribution of linear programs to nodes. Alternatively, methods have been developed where a common data structure is maintained and all nodes access this data structure to obtain a relaxation that requires solution, for example [18]. For a discussion of parallel branch-and-cut algorithms, see [7, 27]. It is also possible to generate cutting planes in parallel; see, for example, [14]. For more information on the use of parallel computers, see the entry on **parallel mixed integer programming**.

Conclusions.

Branch-and-cut methods have been successfully used to solve both specialized integer programming problems such as the traveling salesman problem and vehicle scheduling, and also general integer programming problems. In both cases, these methods are the most promising techniques available for proving optimality. For specialized problems, cutting planes are derived using the polyhedral theory of the underlying problem. For general mixed integer linear programming problems, important components of an efficient implementation include preprocessing, primal heuristics, routines for generating cutting planes such as lift-and-project or Gomory's rounding procedure or cuts derived from knapsack problems, and also routines for lifting constraints to strengthen them. This is an active research area, with refinements and developments being continuously discovered.

References

[1] Applegate, D., Bixby, R., Chvátal, V., and Cook, W.: Finding Cuts in the TSP (A preliminary report), Tech. rep., Mathematics, AT&T Bell Laboratories, Murray Hill, NJ, 1994.

[2] Balas, E.: 'Intersection cuts — a new type of cutting planes for integer programming', *Operations Research* **19** (1971), 19–39.

[3] Balas, E., Ceria, S., and Cornuéjols, G.: 'A lift-and-project cutting plane algorithm for mixed 0-1 programs', *Mathematical Programming* **58** (1993), 295–324.

[4] Balas, E., Ceria, S., and Cornuéjols, G.: 'Mixed 0–1 programming by lift-and-project in a

interior point methods
quadratic assignment problem
parallel computer
parallel mixed integer programming

branch-and-cut framework', *Management Science* **42** (1996), 1229–1246.

[5] Balas, E., Ceria, S., Cornuéjols, G., and Natraj, N.: 'Gomory cuts revisited', *Operations Research Letters* **19** (1996), 1–9.

[6] Bixby, R. E., Ceria, S., McZeal, C. M., and Savelsbergh, M. W. P.: 'An updated mixed integer programming library: MIPLIB 3.0', *Optima* **58** (1998), 12–15, Problems available at *http://www.caam.rice.edu/bixby/miplib/miplib.html*.

[7] Bixby, R. E., Cook, W., Cox, A., and Lee, E. K.: Parallel mixed integer programming, Tech. rep., Department of Computational and Applied Mathematics, Rice University, Houston, Texas, June 1995.

[8] Bixby, R. E., and Lee, E. K.: 'Solving a truck dispatching scheduling problem using branch-and-cut', *Operations Research* **46** (1998), 355–367.

[9] Boyd, E. A.: 'Fenchel Cutting Planes for Integer Programs', *Operations Research* **42** (1994), 53–64.

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

[11] Ceria, S., Cordier, C., Marchand, H., and Wolsey, L. A.: 'Cutting plane algorithms for integer programs with general integer variables', *Mathematical Programming* **81** (1998), 201–214.

[12] Ceria, S., Cornuéjols, G., and Dawande, M.: 'Combining and strengthening Gomory cuts', in E. Balas and J. Clausen (eds.): *Lecture Notes in Computer Science*, Vol. 920, Springer-Verlag, 1995.

[13] Christof, T., Jünger, M., Kececioglu, J., Mutzel, P., and Reinelt, G.: 'A branch-and-cut approach to physical mapping with end-probes': *Proceedings 1st International Conference on Computational Molecular Biology (RECOMB-1), Santa Fe, New Mexico*, 1997.

[14] Christof, T., and Reinelt, G.: Parallel Cutting Plane Generation for the TSP (Extended Abstract), Tech. rep., IWR Heidelberg, Germany, 1995.

[15] Chvátal, V.: 'Edmonds polytopes and a hierarchy of combinatorial problems', *Discrete Mathematics* **4** (1973), 305–337.

[16] Cordier, C., Marchand, H., Laundy, R., and Wolsey, L. A.: bc-opt: A branch-and-cut code for mixed integer programs, Tech. rep., CORE, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, October 1997.

[17] Crowder, H. P., Johnson, E. L., and Padberg, M.: 'Solving large-scale zero-one linear programming problems', *Operations Research* **31** (1983), 803–834.

[18] Eckstein, J.: 'How much communication does parallel branch and bound need?', *INFORMS Journal on Computing* **9** (1997), 15–29.

[19] Gomory, R. E.: 'Outline of an algorithm for integer solutions to linear programs', *Bulletin of the American Mathematical Society* **64** (1958), 275–278.

[20] Gomory, R. E.: 'An algorithm for integer solutions to linear programs', *Recent Advances in Mathematical Programming*, in R. L. Graves and P. Wolfe (eds.). McGraw-Hill, New York, 1963, pp. 269–302.

[21] Grötschel, M., and Holland, O.: 'Solution of large-scale travelling salesman problems', *Mathematical Programming* **51(2)** (1991), 141–202.

[22] Hoffman, K. L., and Padberg, M.: 'Improving LP-Representation of Zero-One Linear Programs for Branch-and-Cut', *ORSA Journal on Computing* **3(2)** (1991), 121–134.

[23] Hoffman, K. L., and Padberg, M.: 'Solving Airline Crew Scheduling Problems by Branch-and-Cut', *Management Science* **39(6)** (1993), 657–682.

[24] Joy, S., Borchers, B., and Mitchell, J. E.: 'A branch-and-cut algorithm for MAX-SAT and weighted MAX-SAT', *Satisfiability Problem: Theory and Applications*, in D. Du, J. Gu, and P. Pardalos (eds.), Vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS/DIMACS, 1997, pp. 519–536.

[25] Jünger, M., and Kaibel, V.: A basic study of the QAP polytope, Tech. Rep. 96.215, Institut für Informatik, Universität zu Köln, Pohligstraße 1, D-50969 Köln, Germany, 1996.

[26] Jünger, M., and Mutzel, P.: 'Maximum planar subgraphs and nice embeddings — practical layout tools', *Algorithmica* **16** (1996), 33–59.

[27] Jünger, M., and Störmer, P.: Solving large-scale traveling salesman problems with parallel branch-and-cut, Tech. Rep. 95.191, Institut für Informatik, Universität zu Köln, Pohligstraße 1, D-50969 Köln, Germany, 1995.

[28] Jünger, M., and Thienel, S.: The design of the branch-and-cut system ABACUS, Tech. Rep. 97.260, Institut für Informatik, Universität zu Köln, Pohligstraße 1, D-50969 Köln, Germany, 1997.

[29] Lovász, L., and Schrijver, A.: 'Cones of matrices and set-functions and 0-1 optimization', *SIAM Journal on Optimization* **1** (1991), 166–190.

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

[31] Nemhauser, G. L., and Wolsey, L. A.: *Integer and Combinatorial Optimization*, John Wiley, New York, 1988.

[32] Padberg, M., and Rinaldi, G.: 'A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems', *SIAM Review* **33(1)** (1991), 60–100.

[33] Sherali, H. D., and Adams, W. P.: 'A hierarchy of relaxation between the continuous and convex hull representations for zero-one programming problems', *SIAM Journal Discrete Mathematics* **3** (1990), 411–430.

[34] Wolsey, L. A.: 'Valid inequalities for mixed integer programs with generalized and variable upper bound constraints', *Discrete Applied Mathematics* **25** (1990), 251–261.

John E. Mitchell
Mathematical Sciences
Rensselaer Polytechnic Institute
Troy NY
USA
E-mail address: mitchj@rpi.edu

AMS 1991 Subject Classification: 90C10, 90C11, 49M35, 90C06.

Key words and phrases: cutting planes, branch-and-bound, integer programs, exact algorithms.