To appear in *Mathematical Programming* (2006)
The original article is available at http://www.springerlink.com

Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs

Jagat Patel
Nortel Networks
3500 Carling Avenue
Ottawa, Ontario K2H 8E9
Canada
patelj@nortelnetworks.com

John W. Chinneck
Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada
chinneck@sce.carleton.ca

June 28, 2006

Abstract

The selection of the branching variable can greatly affect the speed of the branch and bound solution of a mixed-integer or integer linear program. Traditional approaches to branching variable selection rely on estimating the effect of the candidate variables on the objective function. We present a new approach that relies on estimating the impact of the candidate variables on the active constraints in the current LP relaxation. We apply this method to the problem of finding the first feasible solution as quickly as possible. Empirical experiments demonstrate a significant improvement compared to a state-of-the art commercial MIP solver.

1. Introduction

The well-known branch and bound method is the main algorithm for solving mixed-integer, integer, and binary linear programming problems (here referred to collectively as MIP problems). It has a long history, dating to the 1960s [Land and Doig 1960] and has been extensively developed since then (e.g. Johnson et al. [2000]). The general steps of the method, summarized in Algorithm 1, are fairly standard, but there are numerous variations in the details.

A critical element of a successful branch and bound approach is the ability to find a feasible solution quickly. In some cases, a feasible solution is the only goal. Where optimality is needed, finding a feasible incumbent solution quickly permits early pruning and hence the development of a smaller search tree. In very difficult models that may terminate before finding the optimum solution, finding a feasible solution early increases the likelihood that the solver will at least be able to report a usable solution. Finally, some methods for analyzing infeasibility in MIPs

require the repeated solution of variations of the original MIP in which only the feasibility status of the variant MIP is required [Guieu and Chinneck 1999]; finding a feasible solution quickly

Input: mixed-integer program. Incumbent solution = ϕ. List of unexplored nodes = ϕ.

1. Root node is the original model. Solve the LP relaxation of the root node. If LP relaxation is infeasible, exit with infeasible outcome. If LP relaxation is integer-feasible, exit with relaxation solution as optimum.
2. Choose a candidate variable in the current node for branching.
3. Create two child nodes from the current node by branching on the selected variable and add these new nodes to the list of unexplored nodes.
4. If list of unexplored nodes is empty then:
   4.1. If incumbent = ϕ, then exit with "infeasible" outcome.
   4.2. Optimum is incumbent solution: exit with "optimal" outcome.
5. Choose a node from the list of unexplored nodes for expansion.
6. Solve the LP relaxation for the chosen node.
   6.1. If LP relaxation is infeasible then discard the node and go to Step 4.
   6.2. If LP relaxation is feasible and integer-feasible then:
      6.2.1. If LP relaxation objective function value is better than incumbent objective function value then replace incumbent with this solution.
      6.2.2. Go to Step 4.
   6.3. Go to Step 2.

Algorithm 1: General steps in the branch and bound method for solving MIPs.

terminates the assessment, thereby speeding the analysis. For these reasons, we concentrate here on developing faster methods for finding the first feasible solution in MIPs.

Two of the most important aspects of the method are the selection of the next node for expansion (Step 5), and selection of the branching variable (Step 2). Both can have a significant impact on the speed of the solution. After solving the LP relaxation associated with the chosen node in Step 6, the list of *candidate variables* for branching is known: it consists of the integer variables that do not have integer values at the optimum solution of the LP relaxation. In Step 2, one of the candidate variables is chosen for branching, thereby creating two new child nodes. Each child node is created by adding a new variable bound to the model in the parent node. For example, if some variable x_i is chosen for branching, then it must be an integer variable that has a non-integer value f in the LP relaxation solution of the parent LP, i.e. $k_L < f < k_U$, where k_L is the first integer below f and k_U is the first integer above f. One child node is created by adding the variable bound $x_i \leq k_L$ to the model in the parent node, and the other child node is created by adding the variable bound $x_i \geq k_U$ to the model in the parent node.

The most common node selection scheme for solving MIPs via branch and bound is *depth-first*, in which one of the two just-created child nodes is always selected for expansion next (or failing that, the most recently created node). This has the advantage of allowing an immediate advanced start based on the LP relaxation solution for the parent node, thereby increasing the overall speed of solution. There are several common ways to choose between the two child nodes: (i) *branch down*, in which the child node with the added bound $x_i \leq k_L$ is chosen next, (ii) *branch up*, in which the child node with the added bound $x_i \geq k_U$ is chosen next, and (iii) other schemes, e.g. based on whether f is closer to k_L or k_U.

This paper presents a new approach to selecting the branching variable that demonstrates significant improvement over existing state of the art methods in finding the first feasible solution quickly. Changing the policy for branching variable selection can have a dramatic effect on the speed to first feasible solution. For example, for the MIPLIB2003 *momentum1* model, Cplex 9.0 with all default heuristics turned on times out after 28,800 seconds, while Method B described herein reaches a feasible node in just 67 nodes and 74.61 seconds.

Most existing branching variable selection methods estimate the impact of the candidate variables on the objective function in various ways. The candidate variable having the greatest estimated impact is then chosen for branching. In contrast, the new methods developed here recognize that the solution point in an LP relaxation is fixed by the constraints that are active at the optimum. Therefore to cause the greatest movement of the optimum point in the child nodes, choose the candidate variable that has the most impact on the active constraints in the parent node, rather than looking at the impact on the objective function. The general idea is to arrive at child node relaxation optima that are as far apart as possible, in the hopes that one of the child nodes will never be expanded. The "active constraint variable selection methods" developed here use a variety of means to estimate the impact of each candidate variable on the active constraints.

A brief summary of existing branching variable selection schemes follows. In the methods presented by Linderoth and Savelsbergh [1999], Dakin [1965], Benichou et al. [1971], Gauthier and Ribiere [1977] and Eckstein [1994], the idea is to select the branching variable that maximizes the degradation of the objective function value at the optimal solution of the child node LP relaxation. This gives a tighter bound on the unsolved nodes. As pointed out by Linderoth and Savelsbergh [1999], most branching variable selection methods either estimate degradation in the objective function value of the LP relaxation or provide bounds on the degradation. Many estimation methods are based on pseudo-costs introduced by Benichou et al. [1971].

Pseudo-costs estimate the change in the objective function value of an LP per unit change in the value of an integer variable. To compute the pseudo-costs of a variable exactly, both child node LP relaxations must be solved, but this is obviously inefficient when there are many candidate variables, so most methods try to estimate pseudo-costs by solving a small number of extra LP relaxations. An important observation by Benichou et al. [1971] and later reconfirmed by Linderoth and Savelsbergh [1999] is that the pseudo-costs of the integer variables in a particular branch direction remain constant throughout the branch and bound tree with the exception of only a few nodes. This means that once the pseudo-cost of a variable is computed it can be used throughout the B&B tree without having to re-compute it at other nodes.

There are numerous variations on the theme of pseudo-costs. Gauthier and Ribiere [1977] developed "automatic ordered branching" in which the integer variables are sorted in decreasing order of estimated degradation in the objective function value, based on calculations using the pseudo-costs of the variables. Eckstein [1994] keeps track of the pseudo-cost of the integer variables on up and down branches. The average of these values is used for initializing the pseudo-cost of an integer variable that has never been branched on. Forest et al. [1974] suggested

that one way to update the pseudo-cost of the integer variables is to use the last observed value. Linderoth and Savelsbergh [1999] suggested that candidate variables that have never been branched on before should have their pseudo-costs explicitly computed. Both Gauthier and Ribiere [1977] and Linderoth and Savelsbergh [1999] suggested setting a limit on the number of simplex iterations performed for explicitly computing pseudo-costs.

A number of other objective-oriented approaches have been developed. Dakin [1965] and Small [1965] suggested selecting the candidate variable that gives the largest degradation in the objective function value of the LP relaxation problem during the first dual simplex pivot. Beale [1968] uses a similar technique. Tomlin [1969] extended this method to also consider the non-basic integer variables. "Strong branching" (attributed to Bixby by Linderoth and Savelsbergh [1999]) performs a number of dual simplex pivots to get a better lower bound on the degradation in the objective function value at the LP relaxation optimal solution of the child node. Padberg and Rinaldi [1991] used a combination of the objective function coefficient and the fractional part of the candidate variable in their branch-and-cut algorithm for solving large-scale traveling salesman problems.

Branching variable selection can also be based on "Special Ordered Sets", as introduced by Beale and Tomlin [1970]. These are also discussed in Beale and Tomlin [1970] and Linderoth and Savelsbergh [1999].

We now turn our attention to methods that are specifically designed to find feasible solutions in mixed-integer linear programs. Fischetti, Glover and Lodi [2005] recently proposed the Feasibility Pump heuristic for finding a feasible solution to MIP problems without branch and bound. The method alternates between LP-relaxations (which satisfy the linear constraints) and "nearby" integer-feasible roundings of the LP-relaxation solutions (which satisfy the integrality restrictions). The authors report very good results on binary MIP problems. The Feasibility Pump heuristic could provide a useful root node heuristic for reaching feasibility, but the methods developed in this paper will still be valuable when the root node heuristics fail.

Danna, Rothberg and Le Pape [2005] recently introduced the Relaxation Induced Neighbourhood Search (RINS) and guided dives as ways of improving the speed to optimality in solving MIPs. These methods are now included in Cplex 9.0. RINS involves exploring the neighbourhood of the incumbent solution in an effort to find better feasible solutions. The RINS and guided dive methods are complementary to those developed here; it may be profitable to explore using them together.

Balas et al [2001] developed the OCTANE heuristic for generating feasible solutions for pure binary programs within a branch-and-cut framework. The heuristic uses an n-dimensional octagon circumscribing the n-dimensional cube to associate facets with binary solutions. Directions are generated from LP-relaxation solutions, which cross the extended facets of the octagon, and based on which facets are crossed, heuristic solutions are proposed. The authors report very promising heuristic results. Where possible we have compared their empirical results with ours (using node counts as the metric due to differences in machines and software). The results for our methods are very similar to theirs over the few MIPLIB 3.0 models for which they

report node counts. However our methods are also applicable to general integer and mixed-integer programs.

The pivot-and-complement procedure [Balas and Martin 1980] and its descendent pivot-and-shift [Balas and Martin 1986, Balas et al 2004] also focus on obtaining feasible solutions quickly. The more recent pivot-and-shift is a phased rounding procedure. The search phase runs through a cycle of rounding and pivot or shift procedures such as pivoting out basic integer variables, reducing the number of basic integer variables, improving the objective without increasing integer infeasibility, and reducing integer infeasibility. Small neighbourhood searches are also used. During the improvement phase, integer variables are fixed to their best values based on reduced costs, then shifts are applied, and finally a large neighbourhood search is used.

Balas et al [2001] compare OCTANE with several variants of pivot-and-shift and conclude that OCTANE is a competitive alternative. The basis for comparison is the number of branch and bound nodes. OCTANE is faster than the pivot-and-shift variants on 8 of the tested models, but worse on 4. The reported data permits a cursory comparison with the methods developed in this paper since a few of the models are also tested here. Our results are very close to the best result reported for OCTANE or pivot-and-shift in all cases. In one case, Method A developed here is remarkably better: it uses just 6 nodes to reach feasibility for the *misc07* model while OCTANE uses 7504 and all of the pivot-and-shift variants time out. More importantly, the active constraint methods developed here are complementary to OCTANE and pivot-and-shift. The active constraint methods are branching variable selection heuristics and hence can be used whenever the other heuristics are not running.

Though the active constraint methods described here were developed independently, it turns out that they are directly related to the concept of *surrogate constraints* due to Glover et al [Glover 1968, Glover 2003, Lokketangen and Glover 1997]. In the most basic form, a surrogate constraint is any linear combination of a set of linear constraints. When the constraints are all inequalities, their linear combination yields a single linear knapsack inequality. This gives a heuristic method for solving the problem by observing the ratio between the objective function coefficient and the constraint coefficient for each variable (the "bang for the buck"): variable values are selected according to their "bang for the buck" ordering. Various weightings of the individual constraints can be used in constructing the linear combination. Numerous sophisticated methods for selecting the weightings and applying the heuristic have been developed.

As shown later, the active constraint schemes have three main steps: (1) normalization of the active constraints (e.g. by dividing through by the number of candidate variables that appear in the constraint), (2) assigning a "weight" to each candidate variable (e.g. by summing the normalized coefficients for the variable across all of the active constraints), and (3) selecting the variable with the highest total weight as the branching variable. Steps (1) and (2) amount to the creation of a surrogate constraint, however there are significant differences from previous work on surrogate constraints. The most important new development is the application of surrogate constraints to select the branching variable. Other unusual features include *(i)* restricting attention to only the active constraints, *(ii)* restricting attention to only a subset of the variables, and *(iii)* novel normalizations involving absolute values and other innovations.

2. Active Constraint Branching Variable Selection Schemes

As generally defined [Greenberg 2005], the set of "active constraints" includes all equality constraints and all inequalities that hold with equality at the current point; this is the definition used here. Note especially that this definition means that all tight inequalities are included among the active constraints, including both those associated with nonbasic variables, and those that are tight due to degeneracy. The point in question is the optimum point for the LP relaxation associated with a node in the branch and bound tree.

The goal of the schemes described below is to select the branching variable so that the LP relaxation optimum points for the two child nodes are as far apart as possible (in the sense of Euclidean distance). The hope is that these two child node optima will yield such significantly different solutions that one of them will be quite good and lead toward an integer-feasible solution if explored further, while the other child node is so poor that it will never be revisited. This is accomplished by choosing the variable that most affects the active constraints at the parent node LP relaxation optimum. We estimate the impact that an individual candidate variable has on the active constraints by looking at two components: (i) how much influence the variable has within a particular active constraint, and (ii) how much a particular active constraint can be influenced by a single variable.

Measures of the influence of a variable within an active constraint include:

- simple presence of a candidate variable in an active constraint,
- magnitude of the coefficient of a candidate variable in an active constraint, and
- magnitude of the coefficient of a candidate variable in an active constraint normalized by the sum of the magnitudes of all of the coefficients in the active constraint (or the sums of the magnitudes of the coefficients of just the integer variables, or of just the candidate variables).

Measures of how much an active constraint can be influenced include:

- equal valuation for each active constraint,
- inverse of the sum of the magnitudes of all of the coefficients in the active constraint (or the sums of the magnitudes of the coefficients of just the integer variables, or of just the candidate variables), or
- inverse of the number of variables in the active constraint (or the number of integer variables or the number of candidate variables).

In each scheme, a weight w_{ij} is assigned to candidate variable j in active constraint i, based on some combination of the measures mentioned above. The variable having the highest total weight over all of the active constraints is chosen as the branching variable. Variations on the basic schemes include biasing the weights using the dual costs of the active constraints, looking at the single highest w_{ij} instead of the total weight, and a voting scheme. Ties are broken by selecting the variable with maximum infeasibility (defined as minimum distance from integrality); if still tied, the variable with the lowest Cplex-determined index is chosen.

In the course of the research we developed and tested 20 methods using various combinations of the measures listed above. We present here a subset of 7 of the best-performing methods. These are designated by letters or letter combinations that correspond to the original naming scheme for

compatibility with our full sets of experimental results. Several methods not presented here have comparably good results. Some other omitted methods have inferior overall results, but perform spectacularly well on individual models. The reasons for this behaviour are the subject of ongoing research.

The following MIP example is used throughout this section to illustrate the different schemes:

$$\begin{aligned} \text{maximize} \quad & z = 3y_1 - 4x_1 + y_2 - 2y_3 \\ \text{subject to:} \quad & \text{P: } 8y_1 + y_2 - y_3 \le 9 \\ & \text{Q: } -x_1 + 2y_2 + y_3 \le 5 \\ & \text{R: } 3y_1 + 4x_1 + 2y_2 \le 10 \\ & x_1, y_1, y_2, y_3 \ge 0 \\ & x_1 \text{ real; } y_1, y_2, y_3 \text{ integer} \end{aligned} \tag{1}$$

The LP relaxation optimal solution at the root node of the branch and bound tree is $z(y_1, x_1, y_2, y_3) = z(0.8125, 0, 2.5, 0) = 4.9375$. The candidate branching variables are y_1 and y_2. P and Q are the active constraints at the LP relaxation optimum and their dual costs are 0.375 and 0.3125 respectively.

2.1 Weighting Based on the Number of Active Constraints Involving a Candidate Variable

Scheme A uses a simple count of the number of active constraints in which a candidate variable occurs. For candidate variable j in active constraint i, $w_{ij} = 1$ if the candidate variable appears in the active constraint, and $w_{ij} = 0$ if the candidate variable does not appear in the constraint. The total weight is a simple count of the number of active constraints that the candidate variable appears in. In the example, the weights of the candidate variables are found as follows:

Active constraint *i*	$w_{i(y1)}$	$w_{i(y2)}$
P	1	1
Q	0	1
Total:	*1*	*2*

Variable y_2 has the highest total weight and so is selected as the branching variable.

Scheme B recognizes that constraints are relatively easier or more difficult to influence via a single variable. This effect is estimated by noting the sum of the magnitudes of the coefficients of all of the variables in the active constraint. The weight associated with a particular active constraint, instead of being 1 as in Scheme A, is taken as $1/\sum_j|a_{ij}|$, where the coefficient of variable j in constraint i is a_{ij}. Active constraints with many coefficients of large magnitude thus have lower weights since they are likely less influenced by a single variable. $w_{ij} = 0$ if candidate variable j does not appear in active constraint i.

In the example, the weights of the candidate variables are found as follows:

Active constraint *i*	$\sum_j \lvert a_{ij} \rvert$	$w_{i(y1)}$	$w_{i(y2)}$
P	10	0.1	0.1
Q	4	0	0.25
Total:		*0.1*	*0.35*

Variable y_2 has the highest total weight and so is selected as the branching variable.

2.2 Weighting Based on Polling Across Constraints or Methods

Scheme H looks for the maximum impact of a candidate variable on a *single* active constraint when using any particular method. The variable having the largest weight in an individual active constraint is selected as the branching variable. When applied to scheme M for example, the resulting scheme is designated H_M. The weights of the candidate variables using scheme H_M for the example model are as shown with scheme M below. The highest individual weight is associated with variable y_2 in active constraint Q; hence y_2 is selected as the branching variable. We examine methods H_M and H_O in our experiments. Both of the underlying schemes M and O produce good results on their own, however the scheme M results are omitted for clarity.

2.3 Weighting Using the Number of Variables in an Active Constraint

Scheme L adjusts the relative weight of each active constraint according to the number of variables in the constraint. The idea is that constraints that have many variables are less influenced by changes in a single variable because the other variables may be able to compensate. The weight associated with a particular active constraint is taken as $1/N^I_i$ where N^I_i is the number of integer variables in constraint i. $w_{ij} = 1/N^I_i$ if candidate variable j appears in constraint i and $w_{ij} = 0$ if candidate variable j does not appear in active constraint i.

In the example, Scheme L yields the following weights:

Active constraint *i*	N^I_i	$w_{i(y1)}$	$w_{i(y2)}$
P	3	0.333	0.333
Q	2	0	0.5
Total:		*0.333*	*0.833*

Variable y_2 has the highest total weight and so is selected as the branching variable.

Scheme M is identical to Scheme L but considers only the number of fractional valued integer variables in each active constraint. The weight associated with a particular active constraint is taken as $1/N^F_i$ where N^F_i is the number of integer variables currently fractional in constraint i. $w_{ij} = 1/N^F_i$ if candidate variable j appears in constraint i and $w_{ij} = 0$ if candidate variable j does not appear in active constraint i.

In the example, Scheme M yields the following weights:

Active constraint *i*	N^F_i	$w_{i(y1)}$	$w_{i(y2)}$
P	2	0.5	0.5
Q	1	0	1
Total:		*0.5*	*1.5*

Variable y_2 has the highest total weight and so is selected as the branching variable.

2.4 Weighting Using Coefficients and Number of Variables in an Active Constraint

Scheme O considers both the size of the coefficient associated with a candidate variable in an active constraint and the number of variables. As in other schemes, the underlying idea is that larger coefficients indicate a greater impact on the active constraint while more variables indicate a smaller impact. The weight associated with candidate variable j in active constraint i is $w_{ij} = |a_{ij}|/ N^I_i$ and $w_{ij} = 0$ if candidate variable j does not appear in active constraint i.

In the example, Scheme O yields the following weights:

Active constraint *i*	$\mathbf{N^I}_i$	$w_{i(y1)}$	$w_{i(y2)}$
P	3	2.667	0.333
Q	2	0	1.000
Total:		*2.667*	*1.333*

Variable y_1 has the highest total weight and so is selected as the branching variable.

Scheme P is identical to Scheme O except that it considers only the integer variables currently fractional in the active constraint. The weight associated with candidate variable j in active constraint i is $w_{ij} = |a_{ij}|/ N^F_i$ and $w_{ij} = 0$ if candidate variable j does not appear in active constraint i.

In the example, Scheme P yields the following weights:

Active constraint *i*	$\mathbf{N^F}_i$	$w_{i(y1)}$	$w_{i(y2)}$
P	2	4.000	0.500
Q	1	0	2.000
Total:		*4.000*	*2.500*

Variable y_1 has the highest total weight and so is selected as the branching variable.

3. Experimental Setup

Constructing a complete branch and bound MIP solver requires the specification of the node selection method as well as the branching variable selection method. We undertook a preliminary evaluation of a number of node selection schemes in conjunction with the new active-constraint variable selection schemes. These included standard methods such as depth-first search [Dakin 1965], best-first search [Land and Doig 1960], estimation based on pseudo-costs [Benichou et al. 1971], best projection [Hirst 1969, Mitra 1973], and backtracking [Gauthier and Ribiere 1977]. We also investigated some methods tuned to achieving integer feasibility quickly, including minimum number of candidate variables, minimum sum of the integer infeasibilities, minimum number of constraints containing candidate variables, and minimum ratio of number of integer infeasible active constraints to the total number of active constraints. We concluded that depth-first search is generally preferred for the purposes of

achieving integer-feasibility quickly. Depth-first search is also the default method in many commercial solvers. For these reasons, all of the active-constraint solvers use depth-first node selection.

Depth-first search also requires the specification of the branching direction. The common choices are to branch up, to branch down, or to select the direction that corresponds to the closest integer value for the branching variable. We arbitrarily chose to branch up in all of the active constraint solvers.

3.1 Experiments 1 and 2

As described in detail later, Cplex provides the underlying MIP solver framework on which the new variable selection schemes are built. Two experiments are conducted, differing in terms of which internal Cplex-specific heuristics are activated. The heuristics in modern commercial MIP solvers such as Cplex vary widely, and the details are normally proprietary for reasons of competitive advantage. Heuristics may include root node probing, aggregation, presolving, generating cuts, neighbourhood search, etc.

In Experiment 1, all Cplex heuristics are turned off to achieve a straightforward branch and bound set-up. This provides a level playing field for comparing the effectiveness of the various branching variable selection schemes without the confounding effects of additional heuristics. This is the main experiment for evaluating the worth of the new active constraint variable selection algorithms.

All of the new algorithms are compared to the Cplex default scheme and the Experiment 1 results favour the new schemes by a good margin. However it may be that the new schemes perform especially well on the very models that would be solved quickly by the Cplex internal heuristic methods if they were turned on. Experiment 2 provides a supplemental set of results to test this idea by solving the models with all Cplex parameters at their default settings, which generally turns all user-controllable heuristics on. For Experiment 2 we eliminated all models that are solved at the root node from the test set, and then compared solution speed on the resulting smaller set of more difficult models.

3.2 Implementation

The well-known commercial solver Cplex 9.0 [Ilog 2003a, 2003b, 2003c] was used in the experiments reported below. Cplex was used in two ways (i) as a representative state-of-the-art commercial MIP solver with which to compare our new methods, and (ii) as the basic MIP solver framework upon which our new methods are built.

A C++ program manages the interface between Cplex and our new routines, especially the process of obtaining necessary information via the callback routines in the Cplex callable library [ILOG 2003a, 2003b]. The interface program maps the branch and bound nodes to saved data based on the unique node sequence number allocated by the Cplex MIP solver. The interface also allows the user to choose from among the variable and node selection methods described in this paper as well as from the routines internal to Cplex.

The Cplex parameters are set as follows in every active-constraint solver (see the Cplex manuals [Ilog 2003a, 2003b, 2003b]). The corresponding Cplex parameter is shown in brackets.

- Branching variable selection scheme (CPX_PARAM_VARSEL): For Cplex, the default method. For the active constraint methods, overridden by the callback routines.
- Node selection scheme (CPX_PARAM_NODESEL): For Cplex, the default method, which is similar to a depth first search strategy until the first feasible solution is found and is then closer to a best first search. For the active constraint methods, pure depth-first search in Experiment 1 and Cplex default in Experiment 2.
- MIP Emphasis (CPX_PARAM_MIPEMPHASIS): emphasis is on finding a feasible solution.
- Branch and bound node limit (CPX_PARAM_NODELIM): explained below.
- Time Limit (CPX_PARAM_TILIM): explained below.
- Tree Memory Size Limit (CPX_PARAM_TRELIM): 128 MB.
- Node File Size Limit (CPX_PARAM_NODEFILELIM): 800 MB.
- Compress Node File (CPX_PARAM_NODEFILEIND): node file saved on disk and compressed.
- Logging information in file.
- Logging frequency (CPX_PARAM_MIPINTERVAL): 500 nodes.
- All other parameters are set at their default values.

In Experiment 1, all of the preprocessing and node heuristics in the underlying Cplex solver are turned off by these additional parameter settings:

- Pre-solving (CPX_PARAM_PREIND): off.
- Aggregation (CPX_PARAM_AGGIND): off.
- Root Node Heuristic (CPX_PARAM_HEURISTIC): off.
- Internal Node Heuristic (CPX_PARAM_HEURFREQ): off.
- Cut generation (CPX_PARAM_CLIQUES, CPX_PARAM_COVERS, CPX_PARAM_GUBCOVERS, CPX_PARAM_FLOWCOVERS, CPX_PARAM_IMPLBD): off.

3.3 Premature Termination

To permit the completion of many experiments in a reasonable amount of time, the MIP solutions are terminated prematurely if any one of several conditions is met. Termination for any of these reasons does not imply that the algorithm would not complete successfully if given sufficient time or resources.

The most important of the conditions for premature termination is solution time: a maximum of 28,800 seconds (8 hours) of run-time is allowed. Note that this limit includes the calculation time for selecting the branching variable when an active constraint scheme is used. While necessary for practical reasons, this is unfair to the active constraint methods, as explained next.

Cplex performs internal optimizations at each branch and bound node that adjust the number of variables and constraints, and affect whether a particular variable even appears in a particular constraint. This optimization cannot be turned off. For this reason it is not possible to write efficient schemes for searching the data structures during external callbacks. The active constraint methods use simple top-to-bottom searches for the candidate variables in the constraints, which is very time-consuming. In a good implementation, the time taken by the

simple active constraint calculations would be negligible, but it can take a long time in our test software. Some models are terminated prematurely for this reason when the time limit is exceeded, even though the active constraint solver has actually consumed much less Cplex computation time. We present summary statistics on this effect in the tables of results.

Algorithms may also be terminated prematurely if the number of branch and bound nodes solved is too great. There is a hard limit of 100,000 nodes for any experiment. In addition, there is a relative limit for the active constraint methods, as follows. Models are first solved using Cplex. Let k represent the number of nodes taken by Cplex for the solution. Active constraint methods are prematurely terminated if they solve more than $10(k+1)$ nodes since they are clearly shown to be inferior to Cplex at this point.

As described in Section 3.2, limits are also placed on the tree memory size and the node file size, but these limits are not exceeded in any of the experiments reported here.

3.4 Comparison Criteria

Many researchers [Dakin 1965, Tomlin 1969, Benichou et al. 1971, Gauthier and Ribiere 1977, Linderoth and Savelsbergh 1999, Mittelman 2001] have used one or more of the following metrics to compare the effectiveness of MIP solvers in finding a feasible or optimum solution: number of solved nodes (i.e. number of solved LP relaxation problems), total number of simplex iterations over all solved nodes, or solution time (CPU or clock time in seconds). Each metric has advantages and disadvantages.

We perform two sets of experiments to compare the various methods against each other and against Cplex. We use various metrics to compare the solution speed:

- *Number of solved nodes*. This is the most influential factor in terms of memory consumption, and is also closely related to the total computational effort.
- *Number of simplex iterations*. This is a good measure of total computational effort. As opposed to the node count, the number of simplex iterations accounts for the extra effort involved when the algorithm causes a great deal of backtracking, which requires the solution of a node that is quite different from the last one solved (hence there is less scope for an advanced start).
- *Fewer nodes/iterations than Cplex.* The number of models for which the active constraint method takes fewer nodes or iterations than Cplex. This measure takes Cplex as the state of the art for comparison. Note that this does not count instances in which the active constraint method is equally as fast as Cplex (which does happen when the number of solved nodes is used as the measure of speed).
- *Feasibility success ratio (FSR).* Related to the previous measure, this shows the fraction of the models for which the scheme is strictly faster than Cplex.
- *Number of times within 10% of best.* The difference in speed between solutions (measured in number of nodes or iterations) is often quite minor: fractions of a second or just a few nodes. It is misleading to count only the number of times a method is the absolute fastest when it may have been only marginally slower in several cases. For this reason we give the count of the number of times the speed of a method was within 10% (rounded up to next integer) of the fastest speed achieved by any method, including Cplex, for a model. The choice of 10% is arbitrary.

- *Performance profiles.* Performance profiles, as recommended by Dolan and Moré [2002], generalize the idea given in the previous measure by showing how frequently each method is within some arbitrary ratio of the best performance.
- *Quality success ratio (QSR).* The fraction of the models for which the first feasible solution returned by the active constraints method has an optimality gap that is equal to or smaller than the optimality gap for the first feasible solution returned by Cplex. Details follow below.
- *Times terminated.* To complete a large number of experiments in a practical amount of time, solutions are terminated prematurely for the reasons described in Section 3.3. The fewer forced terminations, the better.

Because a different set of models is terminated prematurely for each solver (including Cplex), it is hard to make direct comparisons of average performance. For this reason we also look at a comparable subset of the models: those for which all of the methods terminate successfully. We apply the following additional metrics to the models in the comparable subset:

- *Average nodes, average iterations.*
- *(Average nodes)/(Cplex average nodes), (average iterations)/(Cplex average iterations).*
- *Average ratio to best*. Applies for both nodes and iterations.

We considered the use of clock time as an additional measure of total computational effort, modified to exclude the time spent on the variable selection algorithm for the reasons described in Section 3.3. Because the pattern of the results is closely similar to those returned by the number of simplex iterations metric, we have omitted the clock time results.

Speed to first feasibility is our interest in this paper. However the methods developed here may also prove useful in speeding the solution to optimality, so we are interested in the closeness to optimality of the solutions they return. Feasible solutions obtained quickly may be quite far from optimality, and hence may be detrimental to reaching optimality rapidly. The *Quality Success Ratio* (*QSR*) measures how frequently the first feasible solution returned by an active constraints method has an optimality gap that is equal to or smaller than the optimality gap returned by Cplex at its first feasible solution. The optimality gap is as reported by Cplex: $(|Z_{lo} - Z_{up}|)/(\varepsilon + |Z_{lo}|)$ where Z_{lo} is the current lower bound, Z_{up} is the current upper bound, and ε is a small tolerance to prevent overflow. The QSR is defined as (number of models for which the method returns an optimality gap that is equal to or smaller than the gap returned by Cplex)/(all models compared).

3.5 Test Models

The active constraint methods are tested using the models in the MIPLIB 2003 library [MIPLIB2003, 2005]. This library is an update of the well-known MIPLIB 3.0 library [Bixby et al., 1998] to include more difficult examples. It includes both pure and mixed integer linear programs. Most of the models are from real-world applications, for example *10teams* deals with assigning teams to time slots in the English football league, and *danoint* and *dano3mip* are telecommunication applications. Basic model statistics appear in Table 1.

The library classifies the models according to their level of difficulty: “easy” models are those that can be solved within an hour by a commercial solver, “medium” models are those that have

actually been solved to optimality, but in more than an hour, and "difficult" models are those that have never been solved to optimality. This designation is also shown in Table 1.

For both Experiments 1 and 2, any models that are prematurely terminated by all of the methods (including Cplex) within the specified time and node limits are eliminated from the test sets. For Experiment 1, two models are eliminated: *momentum3* and *stp3d*, leaving a net 58 models in the test set. For Experiment 2, three models are eliminated for this reason: *momentum2*, *momentum3*, and *stp3d*. All three of these models are in the MIPLIB2003 "difficult" category and are among the largest models in the library in terms of number of row constraints. In addition, we eliminate from Experiment 2 any models that are solved at the root node. This eliminates a further 32 models, leaving a net 25 models.

It is interesting to note that *momentum2* is actually solvable by Cplex without triggering premature termination with all of the default heuristics turned off, but not when they are turned on.

Model	Rows	Cols	Int	Bin	Real	NZ	diff.
10teams	230	2025	0	1800	225	12150	e
a1c1s1	3312	3648	0	192	3456	10178	d
aflow30a	479	842	0	421	421	2091	e
aflow40b	1442	2728	0	1364	1364	6783	d
air04	823	8904	0	8904	0	72965	e
air05	426	7195	0	7195	0	52121	e
arki001	1048	1388	123	415	850	20439	d
atlanta-ip	21732	48738	106	46667	1965	257532	d
cap6000	2176	6000	0	6000	0	48243	e
dano3mip	3202	13873	0	552	13321	79655	d
danoint	664	521	0	56	465	3232	m
disctom	399	10000	0	10000	0	30000	e
ds	656	67732	0	67732	0	1024059	d
fast0507	507	63009	0	63009	0	409349	m
fiber	363	1298	0	1254	44	2944	e
fixnet6	478	878	0	378	500	1756	e
gesa2	1392	1224	168	240	816	5064	e
gesa2-o	1248	1224	336	384	504	3672	e
glass4	396	322	0	302	20	1815	d
harp2	112	2993	0	2993	0	5840	m
liu	2178	1156	0	1089	67	10626	d
manna81	6480	3321	3303	18	0	12960	e
markshare1	6	62	0	50	12	312	m
markshare2	7	74	0	60	14	434	m
mas74	13	151	0	150	1	1706	e
mas76	12	151	0	150	1	1640	e
misc07	212	260	0	259	1	8619	e
mkc	3411	5325	0	5323	2	17038	m
mod011	4480	10958	0	96	10862	22254	e
modglob	291	422	0	98	324	968	e

Table 1a: MIPLIB2003 Model Statistics (part 1).

3.6 Machine

The experiments reported here were carried out on Pentium-4 PC running at 2.4 GHz, with 1 gigabyte of random access memory.

4. Empirical Results

Tables 2 to 8 and Figures 1 to 4 summarize the results of the experiments. Experiment 1 establishes the main result concerning the effectiveness of the active constraint methods in branch and bound. Experiment 2 provides supplemental data about how well the active constraint methods work in conjunction with the Cplex default heuristics. In Tables 4-6 and 8-9, bold indicates the best value in a column.

4.1 Experiment 1

We ran Experiment 1 on 20 active constraint algorithm variants. All performed better than Cplex, but we concentrate here on the 5 variants that give the best results overall: methods A, H_M, H_O, O, and P. The raw results for the number of branch and bound nodes solved for each

Model	Rows	Cols	Int	Bin	Real	NZ	diff.
momentum1	42680	5174	0	2349	2825	103198	d
momentum2	24237	3732	1	1808	1923	349695	d
momentum3	56822	13532	1	6598	6933	949495	d
msc98-ip	15850	21143	53	20237	853	92918	d
mzzv11	9499	10240	251	9989	0	134603	e
mzzv42z	10460	11717	235	11482	0	151261	e
net12	14021	14115	0	1603	12512	80384	d
noswot	182	128	25	75	28	735	m
nsrand-ipx	735	6621	0	6620	1	223261	d
nw04	36	87482	0	87482	0	636666	e
opt1217	64	769	0	768	1	1542	d
p2756	755	2756	0	2756	0	8937	e
pk1	45	86	0	55	31	915	e
pp08a	136	240	0	64	176	480	e
pp08aCUTS	246	240	0	64	176	839	e
protfold	2112	1835	0	1835	0	23491	d
qiu	1192	840	0	48	792	3432	e
rd-rplusc-21	125899	622	0	457	165	852384	d
roll3000	2295	1166	492	246	428	29386	d
rout	291	556	15	300	241	2431	e
set1ch	492	712	0	240	472	1412	e
seymour	4944	1372	0	1372	0	33549	m
sp97ar	1761	14101	0	14101	0	290968	d
stp3d	159488	204880	0	204880	0	662128	d
swath	884	6805	0	6724	81	34965	m
t1717	551	73885	0	73885	0	325689	d
timtab1	171	397	107	64	226	829	m
timtab2	294	675	181	113	381	1482	d
tr12-30	750	1080	0	360	720	2508	m
vpm2	234	378	0	168	210	917	e

Table 1b: MIPLIB2003 Model Statistics (part 2).

model in Experiment 1 are given in Table 2, and the number of simplex iterations in each solution is given in Table 3. Summary statistics of these results are given in Tables 4 and 5.

Tables 4 and 5 show the performance of the algorithms over all 58 models, and also over a subset of 40 models for which all methods completed successfully (*comparable* models). It is suitable to count the number of times certain events occur when looking at all 58 models, but the calculation of average results is reserved for the subset of comparable models.

	All 58 Models				40 Comparable Models		
method	times within 10% of best	fewer nodes than Cplex	FSR	times term. (fewer nodes at time-out)	Avg. nodes:	(avg. nodes)/ (Cplex avg. nodes)	avg. ratio to best
Cplex 9.0	7			**4**	1967.5		58.22
A	30	**47**	**0.810**	7 (2)	149.5	0.076	1.19
H_M	28	45	0.776	8(2)	130.5	0.066	1.18
H_O	35	45	0.776	9 (3)	123.3	0.063	1.47
O	**36**	43	0.741	11 (3)	**116.1**	**0.059**	**1.11**
P	32	44	0.759	10 (2)	156.2	0.079	1.37

Table 4: Summary of Experiment 1 (Number of Nodes).

The active constraint methods are invariably much better than Cplex overall (recall that during Experiment 1, all of the Cplex internal heuristics are turned off). Over all 58 models, method A is the best relative to Cplex, completing with fewer branch and bound nodes than Cplex on 81% of the models and with fewer simplex iterations on 74.1% of the models. The worst active constraint methods in both Tables 4 and 5 are still significantly better than Cplex. All of the active constraint methods are within 10% of the fewest nodes or simplex iterations for a significant fraction of the models: 62% (36/58) of the models for the nodes metric for method O

	All 58 Models				40 Comparable Models		
method	times within 10% of best	fewer itns than Cplex	FSR	times term. (fewer itns at time-out)	Avg. itns:	(avg. itns)/ (Cplex avg. itns) [w/o *disctom*]	avg. ratio to best
Cplex 9.0	12			**4**	55052		14.93
A	**30**	**43**	**0.741**	7 (3)	36484	0.663 [**0.214**]	**1.17**
H_M	28	40	0.690	8(3)	**35173**	**0.639** [0.245]	1.18
H_O	23	40	0.690	9 (3)	117320	2.131 [0.237]	1.48
O	25	37	0.638	11 (4)	117401	2.133 [0.239]	1.38
P	**30**	41	0.707	10 (3)	216100	3.925 [0.232]	1.67

Table 5: Summary of Experiment 1 (Simplex Iterations).

and 52% (30/58) of the models for the simplex iterations metric for methods A and P.

Over the 40 comparable models, method O requires just a small fraction (0.059) of the nodes used by Cplex on average, and the other active constraint methods are in a similar range. The average ratios to the smallest number of nodes for each model are very small for the active constraint methods, with a minimum of 1.11 for method O. In terms of the number of simplex iterations, the averages over the 40 comparable models show that Methods A and H_M are significantly faster than Cplex (about a third fewer iterations), but that the other active constraint methods are slower. In contrast, the average ratios to the smallest number of iterations favour all of the active constraint methods by a large margin. This disparity is due to the skewing effect of the *disctom* model, which requires disproportionately large numbers of simplex iterations for the active constraint methods. When *disctom* is ignored, the (average iterations)/(Cplex average iterations) are in the range 0.214 for method A to 0.245 for method H_M.

The number of models for which a method is within 10% of the fastest method (based on nodes or number of simplex iterations) is shown in Tables 4 and 5. Performance profiles generalize this idea to any arbitrary ratio to the fastest method: see Figures 1 and 2. The ideal profile consists of a single dot in the upper left hand corner of the graph, indicating that the method is the fastest for 100% of the models in the test set. Profiles that come closest to this ideal are preferred. Figure 1 shows that the active constraint methods give much better results than Cplex. For example, in Figure 1 consider the fraction of models for which an algorithm completes within twice the number of nodes taken by the best method: method A solves 78% of the models within this limit while Cplex solves just 33%. The "ratio to best axis" is truncated at 10: Cplex solves 33% of models at ratios above this limit, while just 12% of the models are potentially above this limit for method A (the method A curve truncates at this point since 12% of the

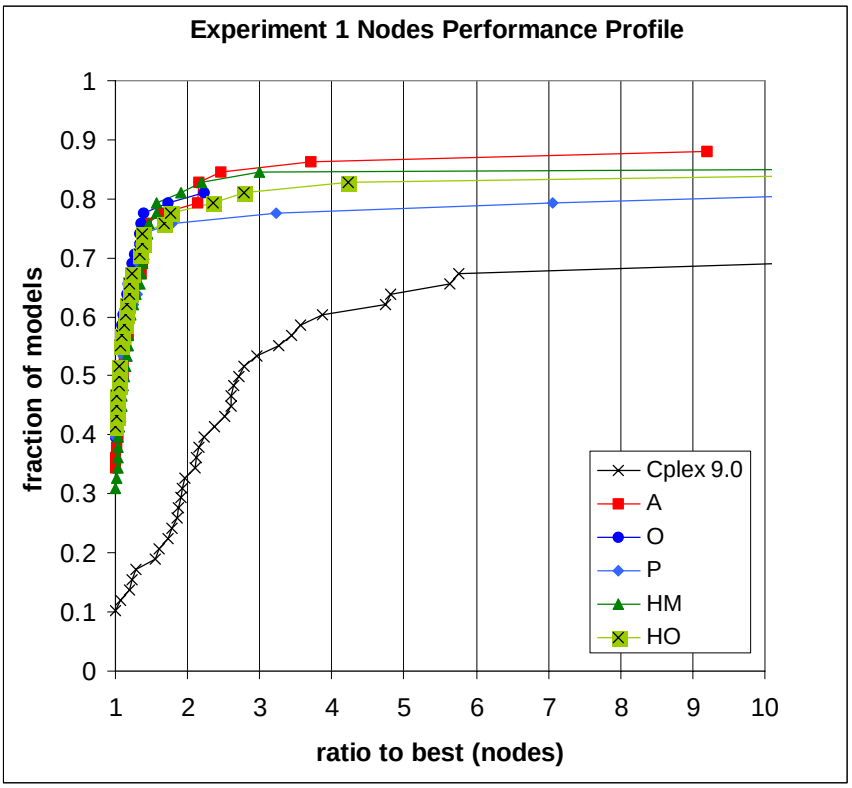

Figure 1: Performance Profiles for Experiment 1 (Nodes).

models are prematurely terminated). The worst ratio to best is recorded by Cplex at 1067 on *aflow3a* for which Cplex solves 23481 nodes while each of the active constraint methods solve just 22 nodes.

The performance profiles based on the number of simplex iterations, given in Figure 2, show the same general story of faster performance by the active constraint methods. Method A again dominates.

While generally much faster, the active constraint methods are terminated prematurely more often than is the case for Cplex. It is not known how many of the models would eventually have been solved to completion if the solver were allowed to run longer. For each active constraint method there are a few models that have used fewer nodes or simplex iterations when terminated by reaching the time limit. As mentioned in Section 3.3, this is likely unfair to the active constraint methods because of their unoptimized data structure searches. The models that experience this problem most frequently are *atlanta-ip, momentum1, msc98-ip* and *rd-plusc-21.* These are exactly the models for which an inefficient search for the candidate variables in the active constraints would be the most time-consuming as shown by the following simple analysis. Sorting the models in descending order of (number of rows) × (number of integer variables + number of binary variables) places these 4 models very close to the top of the list (positions 1, 2, 4 and 6). Note that all four models are rated "difficult", while the models at positions 3 and 5 in the list (*mzzv42z* and *mzzv11*) are rated "easy". Note that if the difficult omitted *stp3d* and *momentum3* were added to this list, they would take positions 1 and 3 in the revised list.

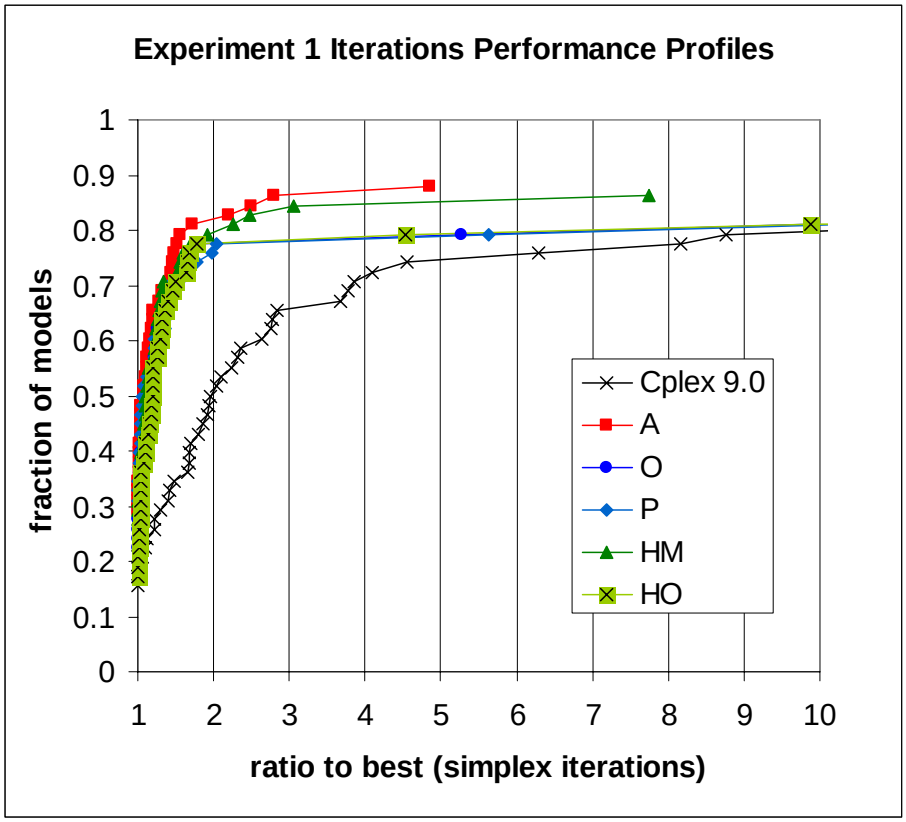

Figure 2: Performance Profiles for Experiment 1 (Simplex Iterations).

The quality success ratio indicates the fraction of models for which an active constraint method has a lower optimality gap at the first feasible solution as compared to the optimality gap at the first feasible solution for Cplex. As shown in Table 6, the quality success ratios for all of the active constraint methods in Experiment 1 are above 50%, and even as high as 78% for method P. This indicates that the active constraint methods are potentially beneficial in the search for optimality since they frequently return a first feasible solution that is closer to optimality than the first feasible solution returned by Cplex.

Overall, Experiment 1 demonstrates that the active constraint methods generally improve the speed of the branch and bound algorithm considerably. The extremely simple method A is surprisingly effective, though a number of other methods are almost as good. Further, the active constraint methods, though seeking feasibility, also improve the movement towards optimality.

Experiment 1 over 40 comparable models	
method	QSR
A	0.53
H_M	0.55
H_O	0.58
O	0.70
P	**0.78**

Experiment 2 over 12 comparable models	
method	QSR
B	**0.75**
H_M	0.50
H_O	0.50
L	0.58
P	0.33

Table 6: Quality Success Ratios.

4.2 Experiment 2

Experiment 2 differs from Experiment 1 in that all of the Cplex internal heuristics are turned on. This greatly complicates the analysis because the exact nature of the interactions between the internal Cplex heuristics on the one hand and the models and active constraint methods for variable selection on the other hand are not known. However, as shown below, speed to first feasibility is generally increased when the active constraint methods are used as compared to the default Cplex variable selection scheme. The results could probably be improved even further if the active constraint variable selection schemes were closely integrated with the heuristics.

Deciding which heuristics to apply is best done by an expert who can customize the choice to the problem at hand. Applying a particular heuristic or combination of heuristics actually slows the generation of the first feasible solution in some cases. In our experiments, the Cplex root node heuristics are quite effective: 32 models are solved by these heuristics and hence are eliminated from the Experiment 2 test set (3 more models are eliminated because all methods are prematurely terminated). For the remaining 25 models, the other heuristics have a mixed effect. Comparing the Cplex results in Experiments 1 and 2, speed to first feasibility is worse in Experiment 2 in about half of the cases, in terms of number of nodes or number of simplex iterations. In some cases the worsening is dramatic: *momentum2* is solved by Cplex in Experiment 1, but is prematurely terminated in Experiment 2; *timtab2* is solved using 14059 nodes in Experiment 1, but 25351 nodes are used in Experiment 2.

Note that the active constraint methods are complementary to the Cplex internal heuristics: they can simply be used whenever a branching variable must be selected. However, because of the unpredictable effects of the heuristics, we take Experiment 1 as the definitive evaluation of the worth of the active constraint methods. The Experiment 2 results are supplementary data that can at least establish whether or not the active constraint methods have a negative impact when combined with the Cplex internal heuristics in a naïve manner.

For clarity, we concentrate on the 5 variants that give the best overall results for Experiment 2: methods B, L, H_M, H_O, and P. Note that methods H_M, H_O, and P are also among the methods that return the best results for Experiment 1. Table 7 shows the number of nodes and the number of simplex iterations used by Cplex and the five active constraint methods in solving each of the 25 models included in Experiment 2. Tables 8 and 9 provide summary statistics on the number of branch and bound nodes and the number of simplex iterations used. Since the test set is small, these statistics are less reliable than those for Experiment 1. This is even truer for the small subset of 12 comparable models.

As shown in Table 8, all of the active constraint methods solve fewer nodes than Cplex on average. For example, over the 25 models, method B is faster than Cplex for 68% of the models, and is terminated prematurely on an additional 20%. Method L gives similar results. The small sample of 12 comparable models shows that the active constraint methods solve only around 20% of the nodes solved by Cplex. The best results are again given by methods B and L. The *disctom* model has a skewing effect on these results since it requires significantly more nodes for

	All 25 Models				12 Comparable Models		
method	times within 10% of best	fewer nodes than Cplex	FSR	times term. (fewer nodes at time-out)	Avg. nodes:	(avg. nodes)/ (Cplex avg. nodes)	avg. ratio to best
Cplex 9.0	4			**1**	1214.6		23.86
B	**9**	**17**	**0.680**	5 (1)	235.0	0.193	2.02
L	7	**17**	**0.680**	6 (1)	**233.0**	**0.192**	2.01
H_M	6	16	0.640	7 (2)	262.9	0.216	2.13
H_O	6	13	0.520	8 (2)	260.9	0.215	1.96
P	**9**	15	0.600	9 (1)	293.8	0.242	**1.27**

Table 8: Summary of Experiment 2 (Number of Nodes).

solution. If *disctom* is ignored, the (average nodes)/(Cplex average nodes) improve significantly for the active constraint methods, ranging from 0.045 for method P to 0.119 for method H_M.

Table 9 summarizes the speed to first feasible solution in terms of the number of simplex iterations. This again shows that the active constraint methods are faster than Cplex on average: the feasibility success ratios are all 56% or higher. The average number of iterations over the small sample of 12 comparable models seems to show that the active constraint methods are worse than Cplex, but this is again due to the skewing effect of *disctom*. When *disctom* is ignored the ratio of the average results favours the active constraint methods by a significant

	All 25 Models				12 Comparable Models		
method	times within 10% of best	fewer itns than Cplex	FSR	times term. (fewer itns at time-out)	Avg. itns	(avg. itns)/ (Cplex avg. itns) [w/o *disctom*]	avg. ratio to best
Cplex 9.0	7			**1**	**32578**		6.89
B	5	14	0.56	5 (1)	400552	12.295 [0.452]	4.37
L	7	14	0.56	6 (2)	400233	12.285 [0.437]	4.38
H_M	2	14	0.56	7 (3)	108898	**3.343** [0.760]	**2.15**
H_O	6	**15**	**0.60**	8 (3)	418697	12.852 [0.785]	4.66
P	**9**	14	0.56	9 (2)	609275	18.702 [**0.367**]	5.90

Table 9: Summary of Experiment 2 (Simplex Iterations).

margin. Over the 12 comparable models, the active constraint methods are faster than Cplex for 75% of the models (methods B, L and H_M) or 83% of the models (methods H_O and P).

Data that compares each method to the best method is included in Tables 8 and 9. The best active constraint methods are within 10% of the best method for 36% (9/25) of the models in terms of number of nodes or number of simplex iterations, better than Cplex. The average ratios to best are also significantly lower than for Cplex. The performance profiles in Figures 3 and 4 give a better picture of the relative performance of the methods.

The nodes performance profiles in Figure 3 show that method P performs particularly well except for the number of times that it is prematurely terminated. Methods B and L are also very good and experience fewer premature terminations. All three active constraint methods are considerably better than Cplex for the models that are not prematurely terminated.

The number of simplex iterations performance profiles in Figure 4 also show that methods B, L and P perform generally better Cplex, but the gap is smaller. The profiles show that these three active constraint methods are always close to the fewest iterations (say within a factor of 2) for the majority of the models, but then suffer from very poor performance (or are prematurely terminated) on the remainder. This suggests that it is important to match the active constraint algorithm to the model at hand.

As discussed in Section 4.1, three models (*momentum1, msc98-ip, rd-plusc-21*) have used fewer simplex iterations than Cplex when they time out for most of the active constraint methods. Again, if we are able to implement an efficient search strategy, the active constraint methods may return better results for these models.

As shown in Table 6, the quality success ratio favours all of the active constraint methods with the exception of method P, and is particularly high for method B. This generally supports the eventual use of active constraint methods in developing algorithms for reaching full optimality more quickly.

The active constraint variable selection methods perform well in Experiment 2, generally increasing the speed to first feasibility in most cases. Methods B, L, and P give the best results, and are the best candidates for further integration with the internal Cplex heuristics in order to yield more consistent results. Method P is interesting in that it is very quick on those models that it completes successfully, but it has a higher rate of premature termination. This again emphasizes the need to carefully select the variable selection scheme based on the characteristics of the model. In the same vein, heuristics must also be carefully selected based on the characteristics of the model. As evidence, consider that its own internal heuristics cause worse results for Cplex itself in about half of the 25 models that survive the root node heuristics.

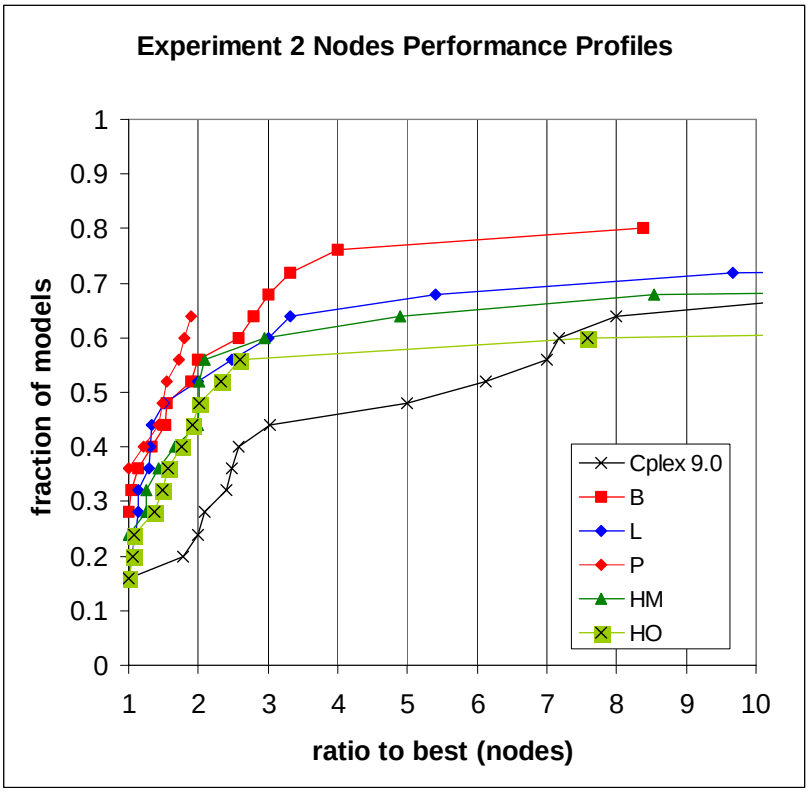

Figure 3: Performance Profiles for Experiment 2 (Number of Nodes).

The supplemental results in Experiment 2 support our contention that the active constraint variable selection methods are useful in branch and bound.

5. Conclusions

Some of the active constraint methods are decidedly superior to a state-of-the-art commercial code in achieving MIP feasibility quickly. Method A clearly dominates in Experiment 1, our main experiment using a straightforward branch and bound setup. The active constraint methods are also generally better when multiple internal heuristics are in operation, as shown by our supplementary Experiment 2, though it is clear that both variable selection algorithms and heuristics must be carefully matched to the characteristics of the model for best results. In the preliminary tests reported here, active constraints method B is preferred over the Cplex internal variable selection scheme when all of the internal heuristics are turned on.

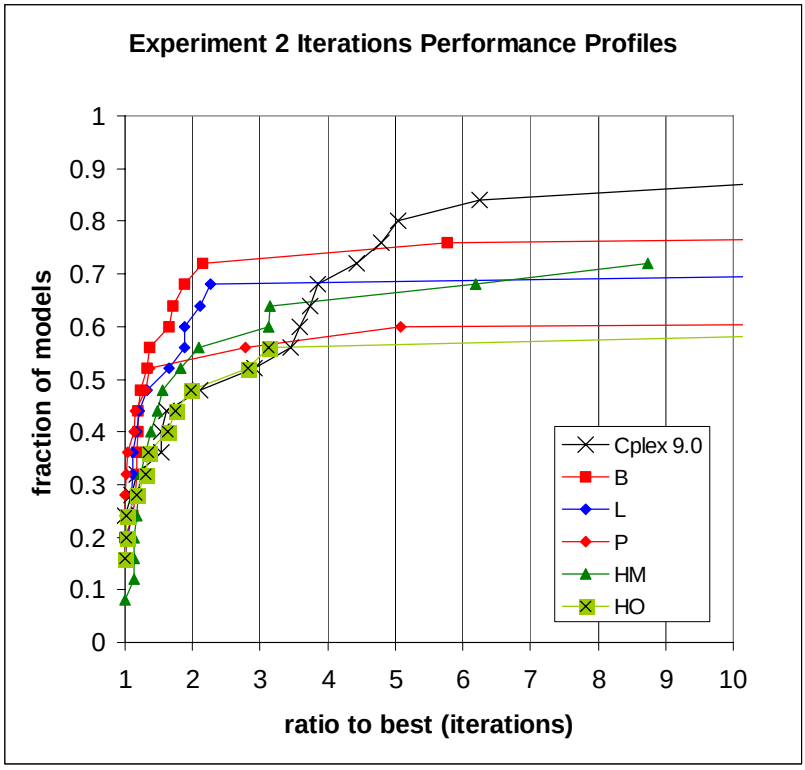

Figure 4: Performance Profiles for Experiment 2 (Simplex Iterations).

Our main conclusion is that the active constraint variable selection methods show significant promise for reducing the time needed to find the first feasible solution. In addition, they may also reduce the time to optimality given that they generally provide a better optimality gap at the first feasible solution as compared to Cplex.

5.1 Future Research

There are many avenues for further research. There are several potential refinements to the current algorithms:

- At present, an active constraints variable selection method is applied straight through from start to finish. It may be better to check the conditions at each node before deciding which method to apply at this particular node. Which method to choose may depend on the number of candidate variables, the number of equality constraints, etc.
- A fuller evaluation of the interaction between the node selection methods and the active constraint variable selection methods should be undertaken.
- Methods for using the active constraint methods to reduce the time to optimality should be studied. There may be ways to combine the active constraint methods with more traditional objective function impact methods to produce a hybrid that is faster to optimality.
- As mentioned in Section 4.2, the exact nature of the interaction between the variable selection scheme and the Cplex internal heuristics should be studied with the goal of making the best matches between the two, and between them and the characteristics of

the model. A larger set of test problems that are not solved by the root node heuristics in Cplex is needed.

- At present the algorithms consider only the active row constraints. It may also be important to consider the active variable bounds, including those introduced by the branch and bound process.

Some implementation issues should also be addressed:

- As noted in several places, our current implementation uses a very inefficient method for searching through the active constraints for candidate variables. Despite the difficulties of dealing with the shifting Cplex data structures, an efficient implementation should be constructed to avoid the problem of spurious premature terminations.
- The time taken by the Cplex heuristics should be measured. For example, Experiment 2 is conducted on the models that did not reach feasibility at the root node when the node heuristics were applied. Those root node heuristics may have taken longer to run than simply solving the entire model using an active constraint method, but this was not measured. Many of the node heuristics involve limited branch and bound operations, and these too could perhaps benefit from the use of the active constraint methods. A detailed study is needed.

The choice of branching direction may have an impact on the results. Preliminary analysis at Ilog shows that branching up during depth-first search may in fact be the best strategy. Further experimentation is needed to gauge the impact of the branching direction in general, and on our experimental results in particular.

Finally, there are promising lines of research suggested by the connection to the concept of surrogate constraints. Some of the new normalizations described in this paper may prove useful in other applications of surrogate constraints. At the same time, some methods known in the surrogate constraints literature may be valuable in selecting branching variables. In addition, when the research moves to the goal of improving speed to optimality, the existing use of surrogate constraints for this purpose may provide useful insights.

Acknowledgements
Thanks are due to Ed Klotz and Ed Rothberg at Ilog Inc. for advice and discussions while working with the Cplex solver. Emilie Danna at Ilog also provided helpful comments on draft versions of the manuscript. Cplex 6.5, 8.0 and 9.0 were provided without charge by Ilog Inc. for the purposes of this study. Christopher Kafka assisted in the later experimental work. The support of NSERC through a Discovery Grant to John W. Chinneck is gratefully acknowledged.

References

E. Balas, S. Ceria, M. Dawande, F. Margot and G. Pataki (2001). "OCTANE: a New Heuristic for Pure 0-1 Programs", *Operations Research* 49, 207-225.

E. Balas and C. Martin (1980). "Pivot and Complement – a Heuristic for 0-1 Programming", *Management Science* 26, 224-234.

E. Balas and C. Martin (1986). “Pivot and Shift – a Heuristic for Mixed Integer Programming”, GSIA, Carnegie Mellon University.

E. Balas, S. Schmieta and C. Wallace (2004). “Pivot and Shift – a Mixed Integer Programming Heuristic”, *Discrete Optimization* 1, 3-12.

E.M.L. Beale (1968). **Mathematical Programming in Practice**, Pitmans, London.

E.M.L. Beale and J.A. Tomlin (1970). “Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables”, *Proceedings of the Fifth International Conference on Operational Research*, Tavistock publication, London, 447-454.

M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent (1971). “Experiments in mixed-integer linear programming”, *Mathematical Programming* 1, 76-94.

R.E. Bixby, S. Ceria, C.M. McZeal, and M.W.P. Savelsbergh (1998). “An Updated Mixed Integer Programming Library: MIPLIB 3.0”, *Optima* 58, 12-15.

R.J. Dakin (1965). “A Tree Search Algorithm for Mixed Integer Programming Problems”, *Computer Journal* 8, 250-255.

E. Danna, E. Rothberg and C. Le Pape (2005). “Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions”, *Mathematical Programming Series A* 102, 71-90.

E.D. Dolan and J. Moré (2002). “Benchmarking Optimization Software with Performance Profiles”, *Mathematical Programming Series A* 91, 201-213.

J. Eckstein (1994). “Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5”, *SIAM Journal on Optimization* 4, 794-814.

M. Fischetti, F. Glover and A. Lodi (2005). “The Feasibility Pump”, *Mathematical Programming Series A* 104, pp. 91-104.

J.J.H. Forest, J.P.H. Hirst, and J.A. Tomlin (1974). “Practical Solution of Large Scale Mixed Integer Programming Problems with UMPIRE”, *Management Science* 20, 736-773.

J.M. Gauthier and G. Ribiere (1977). “Experiments in mixed-integer linear programming”, *Mathematical Programming* 12, 26-47.

F. Glover (1968). “Surrogate Constraints”, *Operations Research* 16, 741-749.

F. Glover (2003). “Tutorial on Surrogate Constraint Approaches for Optimization in Graphs”, *Journal of Heuristics* 9, 175-227.

H.J. Greenberg (2005). "Mathematical Programming Glossary". Online: http://glossary.computing.society.informs.org/

O. Guieu and J.W. Chinneck (1999). "Analyzing Infeasible Mixed-Integer and Integer Linear Programs", *INFORMS Journal on Computing* 11, 63-77.

J.P.H. Hirst (1969). "Features required in Branch-and-Bound Algorithms for (0-1) Mixed Integer Linear Programming", unpublished manuscript.

ILOG Corporation (2003a). **Cplex 9.0 Callable Library Reference Manual**.

ILOG Corporation (2003b). **Cplex 9.0 C++ API Reference Manual**.

ILOG Corporation (2003c). **Cplex 9.0 User's Manual**.

E.L. Johnson, G. L. Nemhauser, and M.W.P. Savelsbergh (2000). "Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition", *INFORMS Journal on Computing* 12, 2-23.

A.H. Land and A.G. Doig (1960). "An Automatic Method for Solving Discrete Programming Problems", *Econometrica* 28, 497-520.

J.T. Linderoth and M.W.P. Savelsbergh (1999). "A computational study of search strategies for Mixed Integer Programming", *INFORMS Journal of Computing* 11, 173-187.

A. Lokketangen and F. Glover (1997). "Surrogate Constraint Analysis – New Heuristics and Learning Schemes for Satisfiability Problems", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 35, 537-572.

G. Mitra (1973). "Investigation of Some Branch-and-Bound Strategies for the Solution of Mixed Integer Linear Programs", *Mathematical Programming* 5, 155-170.

MIPLIB2003 (2005). Library of mixed-integer linear programming test models at http://miplib.zib.de/miplib2003.php.

H. Mittelman (2001). Benchmark for Optimization Software web-site at http://plato.la.asu.edu/bench.html.

M.W. Padberg, and G. Rinaldi (1991). "A Branch-and-Cut Algorithm for the Resolution of Large-scale Symmetric Traveling Salesman Problems", *SIAM Review* 33, 60-100.

R.E. Small (1965). "Mixed Integer Programming by a Branch and Bound Technique", *Proc. of the 3rd IFIP Congress* 2, 450-451.

J.A. Tomlin (1969). "An improved Branch-and-Bound Method for Integer Programming", *Operations Research* 19, 1070-1075.

Model	Cplex9	A	H_M	H_O	O	P
10teams	579	19	14	200	19	168
a1c1s1	7835	174	174	174	174	174
aflow30a	23481	22	22	22	22	22
aflow40b	*n:max*	46	46	33	33	33
air04	31	9	13	38	n:320	29
air05	40	26	19	32	33	25
arki001	318	105	*n:3190*	82	*n:3190*	102
atlanta-ip	1247	496	***t:759***	***t:866***	***t:834***	***t:844***
cap6000	477	*n:4780*	*n:4780*	1331	589	*n:4780*
dano3mip	60	48	47	48	48	47
danoint	20	74	44	*n:210*	*n:210*	*n:210*
disctom	9098	3167	2381	1917	1917	3404
ds	1312	24	46	28	28	32
fast0507	14753	26	41	61	58	34
fiber	69	26	36	12	13	16
fixnet6	181	65	65	65	65	65
gesa2-o	171	65	67	66	65	65
gesa2	143	64	67	64	64	66
glass4	7940	62	62	62	55	62
harp2	126	*n:1270*	53	58	58	374
liu	568	666	670	677	658	665
manna81	272	274	272	272	274	274
markshare1	52	23	23	24	20	20
markshare2	42	30	30	27	27	27
mas74	39	21	21	21	21	21
mas76	26	16	16	16	16	15
misc07	25	7	9	7	7	7
mkc	44	41	45	42	55	41
mod011	49	15	15	15	15	15

Model	Cplex9	A	H_M	H_O	O	P
modglob	66	31	31	31	31	31
momentum1	*t:1219*	***t:206***	***t:100***	***t:149***	***t:191***	74
momentum2	236	*t:647*	*t:678*	*t:223*	*t:658*	*t:688*
msc98-ip	1309	*t:2560*	*t:2673*	128	*t:2673*	*t:2601*
mzzv11	157	80	94	*n:1580*	*n:1580*	*n:1580*
mzzv42z	135	52	71	*n:1360*	72	71
net12	1004	432	1545	47	*t:4904*	*t:2503*
noswot	37	65	30	32	32	32
nsrand-ipx	3301	22	18	19	19	20
nw04	50	3	3	4	4	3
opt1217	130	47	48	44	44	44
p2756	429	127	139	110	113	89
pk1	65	25	25	24	24	24
pp08a	56	49	49	47	47	47
pp08aCUTS	56	30	30	29	30	30
protfold	314	62	*n:3150*	*n:3150*	29	1679
qiu	41	23	19	20	21	21
rd-plusc-21	*n:max*	***t:93***	65	***t:87***	***t:85***	***t:85***
roll3000	600	66	64	48	51	n:6010
rout	124	54	30	25	22	29
set1ch	160	117	117	100	100	100
seymour	155	87	105	154	103	96
sp97ar	3553	60	66	76	58	57
swath	45	33	35	33	24	27
t1717	2505	60	71	69	50	65
timtab1	446	234	234	234	234	234
timtab2	14059	*n:max*	*n:max*	*n:max*	*n:max*	*n:max*
tr12-30	*n:max*	322	327	340	341	326
vpm2	700	31	28	28	34	33

Table 2: Experiment 1 Number of Nodes.

t: indicates time limit exceeded at the number of nodes shown. *n:* indicates node limit exceeded at number of nodes shown. *n:max* indicates 100,000 node limit reached. Bold indicates time limit reached while number of nodes smaller than used by Cplex 9.0

Model	Cplex9	A	H_M	H_O	O	P
10teams	64177	8253	7327	33218	8253	41178
a1c1s1	162715	1277	1263	1263	1277	1277
aflow30a	559252	2128	2128	3770	3770	3770
aflow40b	*n:3712351*	10546	10546	14422	14422	14422
air04	9359	6329	6838	10347	*n:133692*	7738
air05	5563	5599	4103	5162	6654	3936
arki001	5566	2014	*n:38008*	2065	*n:165353*	2731
atlanta-ip	1217175	437934	***t:576607***	***t:372971***	***t:718101***	***t:521645***
cap6000	1503	*n:735620*	*n:735620*	43236	2554	*n:579753*
dano3mip	201015	200091	237900	190739	217601	193004
danoint	3384	16431	8384	*n:44454*	*n:40396*	*n:42159*
disctom	432194	1080459	973598	4273380	4273380	8234181
ds	394023	30737	41260	36526	38553	44966
fast0507	64617	11744	13405	17009	16042	10291
fiber	735	559	726	420	379	543
fixnet6	1383	338	338	338	338	338
gesa2	1184	735	715	718	732	739
gesa2-o	1301	574	560	553	559	556
glass4	11734	153	153	153	148	153
harp2	2017	*n:13168*	1649	1714	1714	3268
liu	3572	2121	2171	2221	2168	2202
manna81	3153	3155	3153	3153	3155	3155
markshare1	115	86	86	77	55	55
markshare2	108	93	93	84	77	77
mas74	246	149	160	171	146	179
mas76	140	127	117	129	117	107
misc07	742	402	385	400	534	534
mkc	2686	2669	2889	2693	3213	2793
mod011	9358	5237	5237	5223	5223	5223

Model	Cplex9	A	H_M	H_O	O	P
modglob	673	302	302	302	302	302
momentum1	*t:1452453*	**t:1316982**	**t:1174742**	**t:1215730**	**t:1100447**	7451
momentum2	160202	*t:716951*	*t:696357*	*t:1077297*	*t:692669*	*t:565815*
msc98-ip	5364578	***t:1231963***	***t:331337***	329561	***t:828853***	***t:943085***
mzzv11	98182	114597	167479	*n:1613459*	*n:709333*	*n:548648*
mzzv42z	33916	95022	103738	*n:206200*	178680	69318
net12	484113	125320	971323	*n:18050*	*t:2999970*	*t:4204447*
noswot	145	182	143	181	173	174
nsrand-ipx	8355	416	667	480	469	440
nw04	513	195	296	257	242	195
opt1217	1210	737	489	487	487	428
p2756	1123	323	336	304	324	297
pk1	510	398	398	360	300	300
pp08a	353	211	211	189	189	189
pp08aCUTS	430	353	378	365	396	396
protfold	1279679	158070	*n:2250448*	*n:1543869*	71819	747318
qiu	5619	3968	2767	3580	4024	4024
rd-plusc-21	*n:342232*	***t:543***	1534	***t:1853***	***t:952***	***t:979***
roll3000	16816	5411	3701	4458	3733	*n:116366*
rout	2036	1325	1084	1098	875	896
set1ch	576	636	636	970	970	970
seymour	25542	22836	28867	33530	29983	24590
sp97ar	37557	4599	5861	6926	5493	5389
swath	3876	3762	3498	3762	3610	3692
t1717	188279	61192	61995	61351	64580	51252
timtab1	1856	950	950	950	950	950
timtab2	75607	*n:292205*	*n:292205*	*n:292205*	*n:292205*	*n:292205*
tr12-30	*n:217919*	754	744	752	743	737
vpm2	3417	278	290	290	287	277

Table 3: Experiment 1 Simplex Iterations.

t: indicates time limit exceeded at the number of iterations shown. *n:* indicates node limit exceeded at number of iterations shown. Bold indicates time limit reached while number of iterations less than used by Cplex 9.0

Number of Nodes

Model	Cplex9	B	H_M	H_O	L	P
10teams	617	285	735	653	285	86
aflow30a	250	28	10	10	54	10
aflow40b	280	80	59	35	20	38
air04	36	14	20	22	14	20
air05	138	30	21	26	30	10
arki001	613	n:6140	n:6140	n:6140	130	100
atlanta-ip	240	t:1252	t:1165	t:1300	t:1140	t:1359
danoint	70	10	49	400	428	n:710
disctom	4770	2051	1987	2121	2051	3082
glass4	7450	41	51	61	45	61
misc07	80	8	7	7	8	7
msc98-ip	1390	30	t:3667	t:3587	290	t:3500
momentum1	t:2491	67	**t:1085**	**t:376**	t:4717	120
mzzv11	157	80	52	n:1580	59	n:1580
mzzv42z	71	61	40	n:720	60	n:720
net12	480	60	t:2792	115	t:3337	t:3457
opt1217	90	43	54	46	56	52
protfold	630	n:6310	360	270	10	10
rd-plusc-21	10464	**t:562**	**t:551**	**t:548**	**t:559**	**t:556**
roll3000	150	57	50	41	n:1510	30
rout	10	20	10	10	20	10
t1717	680	40	60	70	40	30
timtab1	868	3640	874	874	434	n:8690
timtab2	25351	n:max	n:max	n:max	n:max	n:max
tr12-30	174	180	141	70	173	120

Simplex Iterations

Model	Cplex9	B	H_M	H_O	L	P
10teams	63242	54611	130451	129651	54611	41410
aflow30a	8949	3819	1778	3107	4013	2435
aflow40b	22669	10301	12610	17087	6054	16836
air04	9738	6064	9487	8144	6064	7972
air05	10221	5384	3960	4645	5384	2852
arki001	12464	*n:93627*	*n:210379*	*n:84237*	3137	2819
atlanta-ip	310975	*t:956847*	*t:1130465*	*t:774937*	*t:1382224*	*t:479003*
danoint	12537	3260	10189	90771	79932	*n:118684*
disctom	126572	4687261	1105886	4816947	4687261	7214344
glass4	16184	320	359	325	319	325
misc07	2588	604	640	631	604	541
momentum1	*t:8272328*	17451	***t:5515866***	***t:4654537***	***t:3195352***	88498
msc98-ip	2710456	237456	***t:638999***	***t:1252588***	499040	***t:1425058***
mzzv11	72315	97945	81597	*n:194797*	79243	*n:223011*
mzzv42z	26275	31480	30880	*n:73955*	28843	*n:42856*
net12	541559	219099	*t:3001431*	186146	*t:2775561*	*t:2347168*
opt1217	764	435	537	363	684	410
protfold	1923737	*n:4318770*	422879	1122198	71289	68384
rd-plusc-21	84516	***t:2442***	***t:13801***	***t:5601***	***t:6217***	***t:9352***
roll3000	8477	6347	6245	5519	*n:49109*	5552
rout	1412	1517	1290	1320	1517	1485
t1717	125746	33316	36780	39702	33316	20148
timtab1	8367	13978	3161	3161	2425	*n:30845*
timtab2	524654	*n:961195*	*n:416533*	*n:416533*	*n:416400*	*n:565598*
tr12-30	2852	2994	2995	2441	2968	2539

Table 7: Experiment 2 Number of Nodes and Simplex Iterations.

t: indicates time limit exceeded at the number of nodes/iterations shown. *n:* indicates node limit exceeded at number of nodes/iterations shown.. *n:max* indicates 100,000 node limit reached. Bold indicates time limit reached while number of nodes/iterations smaller than used by Cplex 9.0