Computational Experience Using Arc Consistency for Mixed 0-1 Linear Programming

Lou Hafer

lou@cs.sfu.ca

CMPT TR 92-09

School of Computing Science

Simon Fraser University

Burnaby, B.C., V5A 1S6

August 1992

The research described in this paper was supported by the Centre for Systems Science and School of Computing Science at Simon Fraser University.

Abstract

This report describes the implementation of a partial arc consistency algorithm in **bonsai**, a program for mixed 0-1 linear programming. The equivalence of the arc consistency algorithm to a stylized form of Chvátal-Gomory inequality is shown for integer variables. Computational experience on a selection of 51 problems from MIPLIB is reported; 34 subproblems are succesfully solved.

Key Words: Mixed-integer linear programming, LP-based branch-and-bound, arc consistency, Chvátal-Gomory inequalities.

1. Introduction

bonsai is a LP-based branch-and-bound code for mixed-integer linear programming (MILP) in binary (0/1) and continuous variables. The implementation is described in detail in [5]; enhancements since that report include the use of **YLP**, an updated version of the **XLP** LP subroutine library, and a number of algorithmic changes which represent intermediate steps on the way to an implementation of intelligent backtracking.

This technical report concentrates on the arc consistency algorithm. The use of arc consistency techniques to enforce integrality and dynamically tighten bounds on variables complements the use of LP relaxations for optimisation. Broadly speaking, the simplex algorithm for linear programming deals efficiently with continuous variables. Because it deals with the constraint system as a whole, it provides a global perspective for optimisation. Fathoming of subproblems can be performed based on bounds for the objective function value and detection of infeasibility. However, the requirement for integrality must be dealt with separately, by fixing variables and/or adding constraints. Some variables can be uniquely fixed to integral values based on penalty and bounds tests; for the remainder, branching is required to examine each possible value.

In contrast, arc consistency algorithms deal naturally with variables with finite, discrete domains (*e.g.*, a bounded integer variable) as well as bounded continuous variables. Because they work by examining individual constraints and variables, they can prune the domains of individual variables and in this sense are stronger than LP. On the other hand, the global perspective is lost, so that arc consistency cannot always detect infeasibility identified by LP.

The strengths of the two techniques complement each other: LP provides a global view and bounds for optimisation, while arc consistency acts as a sort of generalised logical test, enforcing integrality requirements and improving upper and lower bounds on both integer and continuous variables. In particular, arc consistency and up/down penalties provide complementary ways of identifying binary variables which can be fixed to 0 or 1. This technical report will also show that arc consistency represents a systematic way of generating and applying a stylised form of Chvátal-Gomory inequality.

Section 2 describes the partial arc consistency algorithm implemented in **bonsai** in an algorithmic fashion, following the usual presentation used in the constraint logic programming (CLP) community. Section 3 shows the equivalence to C-G constraints. Section 4 gives computational experience for a set of 51 test problems.

2. The Partial Arc Consistency Algorithm

Arc consistency is commonly used in conjunction with backtracking search to find solutions for constraint satisfaction problems [6]. The worst case behaviour of these algorithms can be very bad [4], but under restricted conditions they can be acceptably efficient. MILP problems provide a favourable set of restrictions:

- The domains of variables are either continuous, possibly with upper and lower bounds, or contiguous sequences of integers, possibly with upper and lower bounds.
- The constraints are linear in the variables, hence each constraint is simply monotonic over the region defined by the domains of the variables.

Under these restrictions, it is possible to implement an efficient partial arc consistency algorithm which can propagate a restriction on the domain of one variable to the domains of other variables; [9] contains a more detailed theoretical analysis. The algorithm is termed partial because it does not necessarily eliminate all inconsistent values from the domain of a variable; in particular, it prunes the domain only by restricting the lower and upper bounds of the variable.

The key observation for integrating arc consistency into a branch-and-bound algorithm is to notice that fixing a variable to a value is precisely equivalent to making its lower and upper bounds equal to that value. Integer variables are handled easily by observing that their bounds must be tightened to the next feasible integer value. *E.g.*, if the lower bound of an integer variable is calculated to be .1, it must be set to 1; if the variable is binary, it is effectively fixed at 1.

2.1. The Arc Consistency Algorithm

The partial arc consistency algorithm used by **bonsai** is a very specialised restriction of the algorithm described in [9]. The specialisation is made possible by the fact that all constraints are summations, and the right-hand side of all constraints is constant.

Suppose that the set of variable indices V_i is divided into two sets for an equality constraint i. Let $V_i^+ = \{j \mid a_{ij} > 0\}$ and $V_i^- = \{j \mid a_{ij} < 0\}$. Then constraint i can be written as

$$\sum_{j \in V_i^+} a_{ij} x_j + \sum_{j \in V_i^-} a_{ij} x_j = b_i. \tag{1}$$

If each variable x_j has lower bound l_j and upper bound u_j, the upper and lower bounds on the value of the left-hand side of (1) can be obtained as

$$UB_i = \sum_{j \in V_i^+} a_{ij} u_j + \sum_{j \in V_i^-} a_{ij} l_j \tag{2}$$

$$LB_i = \sum_{j \in V_i^+} a_{ij} l_j + \sum_{j \in V_i^-} a_{ij} u_j. \tag{3}$$

Conversely, it is possible to calculate an upper or lower bound for an individual variable by solving for that variable. Let x_t be the target variable and assume that $a_{it} > 0$; solving (1) for x_t gives

$$x_t = \frac{1}{a_{it}} \Big(b_i - \big(\sum_{j \in V_i^+ \setminus t} a_{ij} x_j + \sum_{j \in V_i^-} a_{ij} x_j \big) \Big). \tag{4}$$

The upper bound for x_t is obtained by minimising the summations,

$$u_t = \frac{1}{a_{it}} \Big(b_i - \big(\sum_{j \in V_i^+ \setminus t} a_{ij} l_j + \sum_{j \in V_i^-} a_{ij} u_j \big) \Big), \tag{5}$$

which simplifies to

$$u_t = \frac{1}{a_{it}} (b_i - (LB_i - a_{it} l_t)). \tag{6}$$

The lower bound for x_t is calculated as

$$l_t = \frac{1}{a_{it}} (b_i - (UB_i - a_{it} u_t)). \tag{7}$$

The equations (6) and (7) are called *isolations* for the variable x_t. A similar analysis can be followed to obtain the isolations for the case where $a_{it} < 0$. For inequalities ($\geq$ and $\leq$) only one bound of x_t can be tightened, depending on the type of constraint and sign of a_{it}.

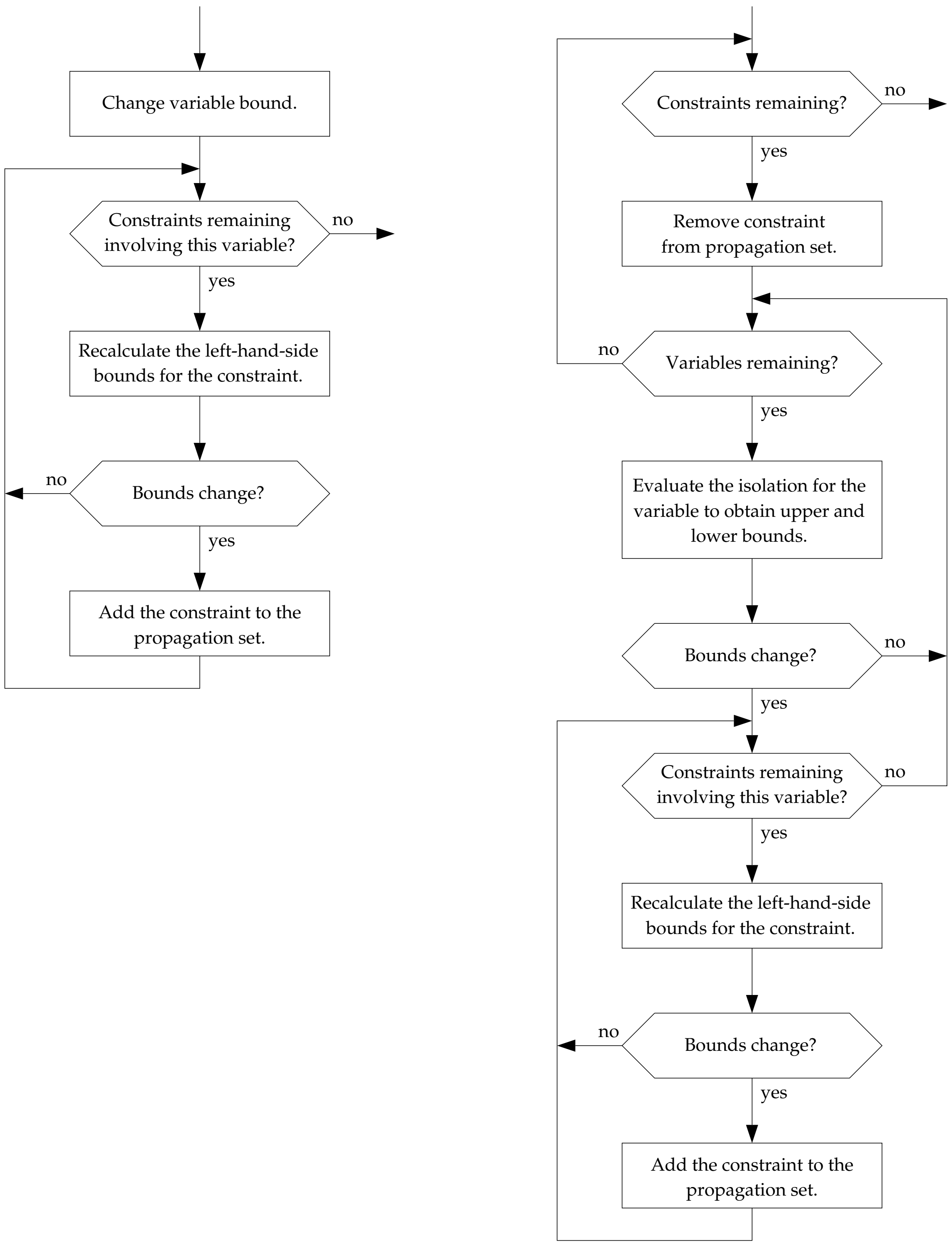

Figure 2-1: Flowchart for the partial arc consistency algorithm

Figure 2-1 shows the operation of the arc consistency algorithm in **bonsai**. It is a two-phase algorithm. An initialisation phase, shown in the flowchart on the left, is run each time **bonsai** fixes variables (*i.e.*, restricts their bounds). The propagation phase, shown in the flowchart on the right, completes the propagation of the bounds changes through the constraint system.

The initialisation phase runs when **bonsai** fixes branching variables and when penalty calculations have identified monotone variables. For each such variable x_j, **bonsai** recalculates the upper and lower bounds UB_i and LB_i for each constraint i where a_{ij} is non-zero. If the bound change should be propagated (*i.e.*, the bound is finite and the change can affect the upper or lower bounds of other variables in the constraint), constraint i is added to the initial set of constraints to be processed.

The propagation phase of the arc consistency algorithm runs once the initial propagation set has been established. A constraint i is removed from the propagation set. The bounds l_t and u_t for each variable x_t in constraint i are evaluated using isolations of the form developed above. If either bound for x_t changes, each constraint k, $k \neq i$, where $a_{kt} \neq 0$ is examined to see if LB_k or UB_k will change in a way that requires further propagation; if so, constraint k is added to the propagation set. The algorithm terminates when the propagation set becomes empty.

Computational experience with this algorithm has shown that the order in which constraints are removed from the propagation set is important to accelerate convergence for problems with continuous variables. **bonsai** keeps the constraints in the propagation set sorted by the total percent change in LB_i and UB_i and always removes the constraint with the greatest change.

It is interesting to note that Brearley, Mitra, and Williams proposed exactly the same set of isolations ([3], p. 62) as part of a preprocessing procedure for LP problems. Their primary goal was the *removal* of redundant bounds on continuous variables, to avoid explicit bounds tests during simplex execution. They mention in passing that in the case of integer variables the benefit of the increased tightness of the bound might outweigh the disadvantage of additional explicit bounds tests.

3. Arc Consistency Viewed as Chvátal-Gomory Inequalities

For $S = \{x \in Z_+^n : Ax \leq b\}$, $A = (a_1, a_2, \ldots, a_n)$, $V = \{1, 2, \ldots, n\}$, Chvátal-Gomory (C-G) inequalities can be generated using the following three-step procedure (*e.g.*, [7], p. 210):

1. Construct a valid constraint as a linear combination of the existing constraints,

$$\sum_{j \in V} (ua_j)x_j \leq ub \qquad u \geq 0.$$

2. Round the left-hand-side coefficients to integer values to construct the valid constraint

$$\sum_{j \in V} \lfloor ua_j \rfloor x_j \leq ub.$$

3. Round the right-hand-side coefficient to an integer value to construct the valid constraint
$$\sum_{j \in V} \lfloor ua_j \rfloor x_j \le \lfloor ub \rfloor.$$

The isolations used by the arc consistency algorithm described in Section 2 are equivalent to a Chvátal-Gomory inequality constructed in the following manner. Consider, for example, constructing an inequality for x_t, where $a_{it} \ge 0$. Combine the constraints

$$a_{it}x_t + \sum_{j \in V^+ \setminus t} a_{ij}x_j + \sum_{j \in V^-} a_{ij}x_j \le b_i \tag{8}$$

$$x_j \le u_j \tag{9}$$

$$-x_j \le -l_j \tag{10}$$

using a vector u where

$u_k = \dfrac{a_{ij}}{a_{it}}$ $\quad k \in (10)$ for $j \in V^+ \setminus t$

$u_k = -\dfrac{a_{ij}}{a_{it}}$ $\quad k \in (9)$ for $j \in V^-$

$u_k = \dfrac{1}{a_{it}}$ $\quad k = i$

$u_k = 0$ $\quad$ otherwise

to produce the inequality
$$x_t \le \frac{b_i}{a_{it}} - \sum_{j \in V^+ \setminus t} \frac{a_{ij}}{a_{it}} l_j - \sum_{j \in V^-} \frac{a_{ij}}{a_{it}} u_j$$

For $x_t \in Z^n$, the constraint can be strengthened to
$$x_t \le \lfloor \frac{b_i}{a_{it}} - \sum_{j \in V^+ \setminus t} \frac{a_{ij}}{a_{it}} l_j - \sum_{j \in V^-} \frac{a_{ij}}{a_{it}} u_j \rfloor$$

An alternative explanation of the operation of the arc consistency algorithm is that it represents the repeated generation and application of this restricted form of C-G inequality. The restricted form of the inequalities means that they do not have to be installed in the constraint system. The strengthening of the constraint system can be carried solely in the upper and lower bounds vectors of the variables.

4. Computational Experience on MIPLIB Problems

The performance of **bonsai** was tested on all problems from MIPLIB [2] which use only continuous and binary variables; the 51 problems are summarised in Table 4-1.

Table 4-2 gives the results for the 34 problems which successfully ran to completion. The data reported are the tour and subproblem where the optimal solution was discovered; the total number of search tree nodes explored; the total number of LP pivots executed, and the total cpu time. The problems were run on a Sun-4/SS2, using unoptimised developmental code with extensive internal consistency checks.

Table 4-3 indicates the apparent reason for failure in the 17 problems which did not complete. The failure of the first six (air04, air05, air06, modglob, vpm1, and misc07) is annoying but not yet serious in an algorithmic sense. The air04, air05, and air 06 models, along with p2756 and p6000 are at the outer boundary of the capacity of computer systems available to the author. The remaining problems are more problematic - they represent instances in which the algorithms used by **bonsai** have difficulty with the problem. There is a possibility that tuning of tolerances in the arc consistency and LP codes may allow **bonsai** to deal with khb05250, rentacar, and mod011; further experimentation will tell the tale.

Example	Constraints	Non-Zeros	Variables			Solution	
			Continuous	Binary	Total	LP	Integer
air01	23	4215	0	771	771	6743.0	6796
air02	50	61555	0	6774	6774	7640.0	7810
air03	124	91028	0	10757	10757	338864.25	340160
air04	823	72965	0	8904	8904	55535.436	56137
air05	426	52121	0	7195	7195	25877.609	26374
air06	825	70806	0	8627	8627	49616.364	49649
bm23	20	505	0	27	27	20.57	34
cracpb1	143	4158	0	572	572	22199.0	22199
diamond	4	9	0	2	2	-1.0	infeas.
egout	98	392	86	55	141	149.589	568.101
enigma	21	289	0	100	100	0.0	0.0
fixnet3	478	1756	500	378	878	40717.018	51973
fixnet4	479	1756	500	378	878	4257.97	8936
fixnet6	479	1756	500	378	878	1200.88	3983
khb05250	101	2700	1326	24	1350	95919464.0	106940226
l152lav	97	9922	0	1989	1989	4656.36	4722

Table 4-1: Characteristics of MIPLIB Problems

Example	Constraints	Non-Zeros	Variables			Solution	
			Continuous	Binary	Total	LP	Integer
lp4l	85	4677	0	1086	1086	2942.5	2967
lseu	28	394	0	89	89	834.68	1120
modglob	291	968	324	98	422	20430947.0	20740508
misc01	54	745	1	82	83	57.0	563.5
misc02	39	413	1	58	59	1010.0	1690
misc03	96	2053	1	159	160	1910.0	3360
misc04	1725	17252	4867	30	4897	2656.42	2666.699
misc05	300	2945	62	74	136	2930.9	2984.5
misc06	820	5859	1696	112	1808	12841.69	12850.8607
misc07	212	8619	1	259	260	1415.0	2810
mod008	6	1243	0	319	319	290.93	307
mod010	146	11203	0	2655	2655	6532.08	6548
mod011	4480	22254	10862	96	10958	-62121982.552	-54558535
mod013	62	192	48	48	96	256.02	280.95
p0033	16	98	0	33	33	2520.57	3089
p0040	23	110	0	40	40	61796.55	62027
p0201	133	1923	0	201	201	6875.0	7615
p0282	241	1966	0	282	282	176867.50	258411
p0291	252	2031	0	291	291	1705.13	5223.7490
p0548	176	1711	0	548	548	315.29	8691
p2756	755	8937	0	2756	2756	2688.75	3124
p6000	2095	??	0	5872	5872	-2351871.325	-2350544
pipex	25	192	0	48	48	773.751	788.263
rentacar	6803	41868	9502	55	9557	28806137.644	30356761
rgn	24	460	80	100	180	48.7999	82.1999
sample2	45	146	46	21	67	247.0	375
sentoy	30	1800	0	60	60	-7839.278	-7772
set1al	493	1412	472	240	712	11145.6	15869.7
set1ch	493	1412	472	240	712	32007.7	54537.7
set1cl	493	1412	472	240	712	1671.96	6484.25
stein9	13	45	0	9	9	4.0	5
stein15	36	120	0	15	15	7.0	9
stein27	118	378	0	27	27	13.0	18
stein45	331	1034	0	45	45	22.0	30
vpm1	234	749	210	168	378	15.4167	20

Table 4-1: Characteristics of MIPLIB Problems

Example	Sol'n found	Nodes	LP Pivots	cpu time
air01	T1:#2	3	248	0:00:10.9
air02	T9:#19	19	1250	0:05:09.4
air03	T1:#3	3	6645	0:05:05.1
bm23	T71:#143	199	1456	0:00:32.2
cracpb1	T34:#68	69	4705	0:04:13.6
diamond	infeas.	3	10	0:00:00.7
egout	T13:#26	953	4926	0:06:00.5
enigma	T8944:#17888	17889	131593	0:21:16.2
l152lav	T712:#1424	1721	66887	0:58:01.9
lp4l	T16:#33	33	4166	0:02:19.2
lseu	T3823:#7646	9867	53911	0:13:23.8
misc01	T162:#324	369	5358	0:01:19.9
misc02	T21:#43	57	798	0:00:08.0
misc03	T274:#548	1277	37998	0:13:08.7
misc04	T8:#17	17	7186	0:29:40.3
misc05	T108:#217	383	10772	0:10:09.5
misc06	T50:#101	115	6343	0:23:49.4
mod008	T1889:#3778	5761	40223	0:13:55.6
mod010	T336:#673	685	26276	0:31:00.8
mod013	T169:#338	417	3671	0:01:01.3
p0033	T24:#49	91	455	0:00:08.1
p0040	T24:#49	49	260	0:00:06.1
p0201	T195:#391	601	15680	0:18:50.2
p0282	T38:#76	139	1918	0:01:24.7
p0291	T19:#38	45	428	0:00:30.0
p0548	T1780:#3560	3661	30368	1:08:47.0
pipex	T638:#1277	1289	5733	0:02:43.8
rgn	T30:#60	2519	17452	0:08:28.1
sample2	T57:#115	141	1024	0:00:12.3
sentoy	T58:#116	277	2238	0:02:05.2
stein9	T4:#9	13	67	0:00:00.6
stein15	T5:#11	39	410	0:00:03.4
stein27	T11:#22	1671	18912	0:05:41.9
stein45	T696:#1393	31855	679865	6:42:20.3

Table 4-2: Results for Test Problems Run to Completion

Table 4-4 provides a comparison of the performance of **bonsai** with the **MINTO** and **LINDO** codes for a much more limited set of test problems. The data for **MINTO** is drawn from [8], while the data for **LINDO** is drawn from [1]. The data reported for **LINDO** did not include execution times. For comparability with the times given in [8], the times reported for **bonsai** in Table 4-4 are for runs executed on a Sun-4/SS1+.

Example	Apparent Reason for Failure
air04, air05, air06	Exceeded an arbitrary limit of 10000 iterations while attempting to solve the initial linear relaxation.
modglob, vpm1, misc07	Aborted due to program bug(s).
khb05250, rentacar, mod011	Report incorrect solutions. Numerical accuracy appears to be at the root of the problem.
set1al, set1ch, set1cl, fixnet3, fixnet4, fixnet6	Terminated after exceeding allowable search limits.
p2756, p6000	Not yet attempted.

Table 4-3: Results for Test Problems Not Run to Completion

Example	**bonsai**		**MINTO**		**MINTO** (-s)		**LINDO**
	Nodes	cpu time	Nodes	cpu time	Nodes	cpu time	Nodes
DIAMOND	3	00:01	1	00:00	7	00:00	
P0033	91	00:06	5	00:01	8291	01:58	7086
P0040	49	00:04	1	00:01	139	00:03	104
P0201	601	14:37	691	7:01	4900	20:35	1730
P0282	139	02:39					10252
BM23	199	00:24	241	00:41	1978	01:24	
LSEU	9867	21:18	193	00:58	63403	34:02	
EGOUT	953	04:42	13	00:02	70220	36:23	

Table 4-4: Comparison of **bonsai**, **MINTO**, and **LINDO**

MINTO searches for strong cuts — clique and implication inequalities, and lifted knapsack and generalised flow covers — and clearly outperforms **bonsai** in problems where it can find these cuts. **bonsai** does not do badly, however, occasionally outperforming **MINTO** in situations where these cuts are absent or not as effective. **bonsai** compares favourably with **LINDO**, and with **MINTO** when it is run with cuts disabled (the column labelled "**MINTO** (-s)" in Table 4-4).

The data would seem to justify a conclusion that the use of arc consistency is a relatively robust technique with wide applicability. While not as effective as stronger cuts, when those cuts can be generated, arc consistency seems to be effective over a broader range of problems.

References

[1] Balas, E., Ceria, S., Cornuéjols, G.
A Lift-and-Project Cutting Plane Algorithm for Mixed 0-1 Programs.
Management Science Research Report No. #576, Management Science Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213, October, 1991.

[2] Bixby, R., Boyd, A., Indovina, R.
MIPLIB: A Test Set of Mixed Integer Programming Problems.
SIAM News p.16, March, 1992.

[3] Brearley, A., Mitra, G., Williams, H.
Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm.
Mathematical Programming 8(1):54-83, February, 1975.

[4] Davis, E.
Constraint Propagation with Interval Labels.
Artificial Intelligence 32:281-331, 1987.

[5] Hafer, L.
bonsai: Teaching a clown to prune trees.
Technical Report CMPT TR 91-6, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, July, 1991.

[6] Nadel, B.
Constraint Satisfaction Algorithms.
Computational Intelligence 5(4):188-224, November, 1989.

[7] Nemhauser, G., Wolsey, L.
Integer and Combinatorial Optimization.
John Wiley & Sons, New York, 1988.
ISBN 0-471-82819-X.

[8] Savelsbergh, M., Sigismondi, G., Nemhauser, G.
Functional Description of MINTO, a Mixed-INTeger Optimizer.
Technical Report, Computational Optimisation Center, School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, 30332, May, 1992.

[9] Sidebottom, G.
Satisfaction of Constraints on Non-negative Integer Arithmetic Expressions.
Open File Report #1990-15, Alberta Research Council, 6815 8th Street N.E., Calgary, Alberta, T2E 7H7, July, 1990.

Table of Contents

List of Tables

List of Figures