Yoyo search: a bisection cutting-plane method

Matteo Fischetti(*) and Domenico Salvagnin(∘)

(*) *DEI, University of Padova, Italy*
(∘) *DMPA, University of Padova, Italy*
e-mail: matteo.fischetti@unipd.it, salvagni@math.unipd.it

November 20th, 2009

Abstract

Cutting plane methods are widely used for solving convex optimization problems and are of fundamental importance, e.g., to provide tight bounds for Mixed-Integer Programs (MIPs). These methods are made by two equally important components: (i) the separation procedure (oracle) that *produces* the cut(s) used to tighten the current relaxation, and (ii) the overall search framework that actually *uses* the generated cuts and determines the next point to cut. In the last 50 years, a considerable research effort has been devoted to the study of effective families of cutting planes, as well as to the definition of sound separation procedures and selection criteria. However, the search component was much less studied—at least by the MIP community, where the "standard" approach almost invariably consists of cutting an optimal vertex of the current LP relaxation.

In this paper we introduce a new search method that generalizes 1-dimensional binary search and produces *two* convergent trajectories of points—one made by optimal LP vertices as in the standard cutting-plane method, and the other by "internal" points used to produce deeper cuts and updated on the fly with no computational overhead. The method is called *yoyo search* because the point to be cut swings in and out of the oracle's reach. It can be viewed as a simple way to exploit the internal-point information inside a standard cutting plane method, hence its implementation within a branch-and-cut scheme is likely to be less problematic than that using pure interior point methods such as the analytic center one.

Preliminary computational results are presented, showing that the yoyo and the standard methods have comparable performance (in terms of number of cuts generated and computing time) when the separation procedure is able to generate very tight (facet defining) cuts. When shallow cuts are generated, instead, our yoyo search outperforms the standard method, producing much better bounds within significantly shorter computing times.

Keywords: Mixed-integer programming, cutting planes, binary search, ellipsoid and analytic center methods.

1 Introduction

Cutting plane methods are widely used for solving convex optimization problems and are of fundamental importance, e.g., to provide tight bounds for Mixed-Integer Programs (MIPs).

Cutting plane methods are made by two equally important components: (i) the separation procedure (oracle) that *produces* the cut(s) used to tighten the current relaxation, and (ii) the overall search framework that actually *uses* the generated cuts and determines the next point to cut.

In the last 50 years, a considerable research effort has been devoted to the study of effective families of cutting planes, as well as to the definition of sound separation procedures and cut selection criteria [8, 9]. However, the search component was much less studied, at least in the MIP context where one typically cuts a vertex of the current LP relaxation, and then reoptimizes the new LP to get a new vertex to cut (a notable exception is the recent paper [17] dealing with Benders' decomposition). The resulting approach—sometimes called "the Kelley method" [14]—can however be rather inefficient, the main so if the separation procedure is not able to produce strong (e.g., facet defining or, at least, supporting) cuts.

As a matter of fact, alternative search schemes are available that work with non-extreme (internal) points [11, 12, 20], including the famous ellipsoid [6, 19] and analytic center [3, 13, 18] methods; we refer the reader to [7] for an introduction. The convergence behavior of these search methods is less dependant on the quality of the generated cuts, which is a big advantage when working with general MIPs where separation procedures tend to saturate and to produce shallow cuts. A drawback is that, at each iteration, one needs to recompute a certain "core" point, a task that can be significantly more time consuming than a simple LP reoptimization. In addition, these methods are allowed to produce invalid cuts based on the objective function value, that may be problematic to handle in a branch-and-cut context.

In this paper we introduce a hybrid search method, called *yoyo search*, that generates *two* convergent trajectories of points—one made by optimal LP vertices as in the standard method, and the other by "internal" points used to produce deeper cuts and updated on the fly with no computational overhead. The new method can therefore be viewed as a simple way to exploit the internal-point information inside a standard cutting plane method, hence its implementation within branch-and-cut scheme is likely to be less problematic than that using pure internal point methods. The method is described in Section 2.

Computational results are presented in Section 3, showing that the yoyo the standard methods have comparable performance—in terms of number of cuts generated and computing time—when the separation procedure is able to generate very tight (facet defining) cuts. When shallow cuts are generated, instead, our yoyo search outperforms the standard method, in that it produces much tighter bounds within shorter computing times.

2 Yoyo search

Let us consider a MIP of the form

$$\min\{c^T x : Ax \leq b,\ x_j \in \mathbb{Z}\ \forall j \in J\}$$

and let $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ denote the associated LP relaxation polyhedron. In addition, let us assume the oracle structure allows one to define a "cut closure", P_1, obtained by intersecting P with the half-spaces induced by all possible inequalities returned by the oracle. Cutting plane methods are meant to compute $z_1 := \min\{c^T x : x \in P_1\}$, with P_1 described implicitly through the oracle.

Our search method works with two points: an "internal" (possibly non optimal) point $q \in P_1$, and an optimal vertex x^* of P (possibly not in P_1). By construction, the final (unknown) value z_1 belongs to the *uncertainty interval* $[c^T x^*, c^T q]$, i.e., at each iteration both a lower and an upper bound on z_1 are available. If the two points q and x^* coincide, the cutting plane method ends. Otherwise, we apply a bisection step over the line segment $[x^*, q]$, i.e., we invoke the separation procedure in the attempt of cutting the middle point $y := (x^* + q)/2$. If a violated cut is returned, we add it to the current LP that is reoptimized to update x^*, hopefully reducing the current lower bound $c^T x^*$. Otherwise, we update $q := y$, thus improving the upper bound and actually *halving* the current uncertainty interval.

The method is called "yoyo search" because the point y to be cut swings between inside and outside P_1. Note that for 1-dimensional problems ($P \subset \mathbb{R}^1$), yoyo and binary search methods coincide. An outline of this basic scheme is given in Figure 1.

The basic scheme can easily be improved in its final iterations. Indeed, it may happen that x^* already belongs to P_1, but the search is not stopped because the internal point q is still far from x^*. A simple fix is to count the number of consecutive updates to q, say k, and to try and separate directly x^* in case $k > 3$. If the separation is unsuccessful, then we can terminate the search, otherwise we reset counter k and continue with the usual strategy of cutting the middle point y.

A graphical representation of the two different type of iterations arising during the search is given in Figure 2. The algorithm starts with the pair (x_0, q_0). At the first iteration, the separation of the middle point y_0 is unsuccessful, and thus the internal point is updated to q_1. In the second iteration, a cut violated by the middle point y_1 is found and added to P, obtaining the new vertex x_1. Note that in the first iteration the uncertainty interval is halved, but this is not the case in the second iteration, where there is only a small improvement in the lower bound value (for 1-dimensional

input : LP $\equiv \min\{c^T x : x \in P\}$
input : separation procedure `Oracle(y)` implicitly defining P_1
input : internal point $q \in P_1$
output: optimal solution x^* of $\min\{c^T x : x \in P_1\}$

$x^* := \arg\min\{c^T x : x \in P\}$
while $x^* \neq q$ **do**
  $y := (x^* + q)/2$
  if `Oracle(y)` *finds a cut* $\alpha^T x \leq \alpha_0$ *violated by* y **then**
    $P := P \cap \{x \mid \alpha^T x \leq \alpha_0\}$
    $x^* := \arg\min\{c^T x : x \in P\}$
  else
    $q := y$
  end
end

Figure 1: Yoyo search.

problems even in this second case the interval would be halved, and the method would become binary search).

As to the initialization of $q \in P_1$, this is a trivial task in many practical settings. For example, when solving MIPs, any feasible integer solution can be used. Other examples are given in Section 3, where the particular structure of the problems at hand allows for a simple formula to define a suitable q. If all else fails, however, a dedicated phase-1 procedure is required, that works, e.g., along the following lines. Let

$$P' := \{x \in P : x \text{ satisfies all the cuts produced by the oracle so far}\}$$

be the current LP relaxation polyhedron ($P' = P$ at the very beginning). At each iteration, we call a black box solver to find a (hopefully internal) point $q \in P'$ to be passed to the separation oracle, until no cut is generated (hence $q \in P_1$, as required). In our computational experience, we found that applying a fast interior-point algorithm (e.g., IBM-ILOG Cplex barrier) to the problem with null objective function $0^T x$ was typically very effective. Dedicated analytic-center (heuristic) implementations are also possible, that we plan to investigate in a near future.

An important property of yoyo search is its ability to produce both a lower and an upper bound on z_1, thus allowing for early termination, e.g., when the two bounds are close enough, or in branch-and-cut methods when the upper bound is smaller than the incumbent value and there is no hope to fathom the current node.

A main advantage of yoyo search over the standard method is that the point to be cut is typically "well inside" P, so deeper cuts are produced.

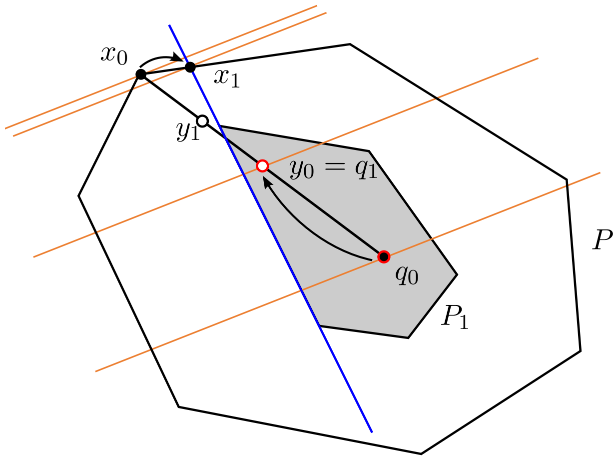

Figure 2: Yoyo search example.

This is confirmed by our computational experience. A possible drawback is that the computing time spent within the separation oracle may be affected by the fact that the point y to be cut can be significantly denser than the LP optimal vertex x^*. In addition, a non-exact (heuristic) separation oracle can interfere with yoyo search, in that a point $y \notin P_1$ not cut by the oracle can erroneously lead to an update of q, hence producing wrong upper bounds, weaker cuts, or even cycling.

With respect to the ellipsoid/analytic-center method, the internal point q of yoyo search needs not to be recomputed a each iteration (a time consuming task), but it is just updated with no overhead in the iterations not producing a violated cut—those are in a sense the "most successful" ones in that they halve the uncertainty about z_1. Another practically very important advantage is that the point q needs not to be in the strict interior of the current LP polyhedron P', i.e., a point lying on a face of P' is allowed (as already mentioned, this can be exploited to provide an initial point q with little computational effort). Note that this property is not shared by the ellipsoid/analytic-center methods, that may need so-called *neutral* cuts that are just tight at the point q to be separated and that must be forced to become slack at the subsequent point q. Finally, whenever $q \in P_1$, the ellipsoid/analytic-center method needs to introduce a (neutral) cut $c^T x \leq c^T q$, that is however invalid for P_1 and hence needs to be removed in a branch-and-cut context.

3 Computational results

To computationally compare the performance of yoyo search with that of the standard (Kelley's) cutting plane method, we performed two kinds of experiments. A comparison with the analytic-center method is planned in the near future.

3.1 Mimicking different cut behaviors

Our first experiments were meant to analyze the yoyo-search performance in a controlled environment where we can tune the quality of the cuts returned by the separation oracle. To this end, we considered the task of solving to proven optimality a given LP problem with bounded variables, namely $\min\{c^T x : l \leq x \leq u, A'x \leq b', A''x = b''\}$, by using a cutting plane scheme. We then define $P := \{x \in \mathbb{R}^n : l \leq x \leq u, A''x = b''\}$, while the list of all the remaining constraints (i.e., the rows of $A'x \leq b'$) is stored inside the separation procedure and can be accessed only through the oracle calls. Given the point $y \in P$ to be separated, the oracle returns a single violated cut $\alpha^T x \leq \alpha_0$ (if any) defined according to one of the following three selection scenarios:

A) select a "deepest" violated cut in the list, i.e., a one that maximizes the Euclidean distance of y from the cut hyperplane;

B) the returned cut is the convex combination (with uniform coefficients) of the deepest one and of the (at most) first 10 violated or tight cuts encountered when scanning the list;

C) the cut is first defined as in case B, and then its right-hand side α_0 is weakened so as to half the degree of violation.

Scenario A simulates the availability of an "almost ideal" separation procedure that is able to detect a deep (typically facet-defining) cut at each call. Scenario B simulates a more realistic scenario where the separation procedure is still able to define reasonably good cuts, though it may be tricked by the presence of alternative violated or tight constraints. Finally, scenario C simulates a common situation where the returned cut is not supporting P_1.

Our testbed is made by the root node relaxation of 19 MIPLIB-2003 [1] instances (namely: a1c1s1, aflow40b, arki001, atlanta-ip, cap6000, dano3mip, gesa2-o, gesa2, liu, manna81, mkc, momentum1, momentum2, msc98-ip, mzzv11, mzzv42z, net12, seymour, and sp97ar) and of 14 set covering instances taken from the ORLIB [4] (scpclr11, scpclr12, scpclr13, scpcyc08, scpnrg1, scpnrg2, scpnrg3, scpnrg4, scpnrg5, scpnrh1, scpnrh2, scpnrh3, scpnrh4, and scpnrh5). These instances were chosen because they have a reasonable number of inequalities, both in absolute terms and with respect to the number of linear equations (if any).

For all instances, our yoyo search was initialized with an internal point q found with the previously-described phase 1 algorithm (the cuts generated during this phase were discarded before moving to the next phase, i.e. they were not passed to the yoyo search, hence the associated computing time is not reported). Note that for the set covering instances in our testbed, the first point $q = (1/2, \cdots, 1/2)$ is always inside P_1 (because each row is covered by at least 2 columns), so no phase 1 iterations were needed.

The outcome of this first set of experiments in reported in Table 1, where at most 10,000 iterations (i.e., calls to the oracle) were allowed. Column %Cl.Gap reports the percentage of gap closed (100% meaning the LP was solved to proven optimality); all computing times are expressed in CPU seconds. As expected, the standard method (std) worked quite well under scenario A, but its performance degraded steeply under scenarios B and C. Yoyo search had an identically good performance under scenario A, but it was much more effective under scenario B (about 10 times faster than std on the set covering instances), and closed considerably more gap under scenario C (on average, about 68% instead of 35% for miplib instances). Tables 2 and 3 give more details on the runs on set covering instances under scenarios B and C, respectively, and also report the number of iterations (cuts) needed to close 90%, 95%, and 99% of the initial gap, respectively.

		itr		time		%Cl.Gap	
testbed	scenario	std	yoyo	std	yoyo	std	yoyo
scp	A	360.2	381.8	7.04	7.60	100.0	100.0
	B	6,248.4	871.5	1,558.59	157.85	100.0	100.0
	C	10,000.0	3,471.0	1,936.75	903.18	92.2	100.0
miplib	A	761.2	738.2	5.37	4.76	100.0	100.0
	B	8,720.0	3,442.1	267.59	144.95	46.0	77.7
	C	10,000.0	6,207.2	171.96	221.87	34.7	68.1

Table 1: Comparing yoyo search (yoyo) and the standard cutting plane method (std) on three different separation scenarios; we report geometric means for iterations and CPU times, and arithmetic means for the closed gap.

3.2 Yoyo search with Benders' cuts

In our second set of experiments, we considered a classical Benders' decomposition approach [5] where the separation oracle solves a certain cut-generating LP (known as the Benders slave). Note that the exploitation of some internal point information, albeit quite different from what presented

problem	method	itr 90%	itr 95%	itr 99%	%Cl.Gap	totTime	totItr
scpclr11	std	917	1,293	2,197	100.0	263.37	2,958
	yoyo	148	195	252	100.0	12.75	381
scpclr12	std	2,267	3,466	5,138	100.0	979.38	5,808
	yoyo	61	96	233	100.0	146.91	474
scpclr13	std	1,883	3,031	5,952	99.9	8,970.94	10,000
	yoyo	93	125	352	100.0	920.21	983
scpnrg1	std	4,205	5,364	6,657	100.0	2,091.78	7,626
	yoyo	602	736	1,009	100.0	238.65	1,283
scpnrg2	std	3,986	4,904	6,294	100.0	1,963.68	7,223
	yoyo	555	734	973	100.0	217.06	1,243
scpnrg3	std	3,256	4,322	5,522	100.0	1,551.54	6,659
	yoyo	465	595	876	100.0	203.06	1,195
scpnrg4	std	4,229	5,126	6,449	100.0	1,651.24	7,475
	yoyo	568	687	959	100.0	254.43	1,274
scpnrg5	std	4,072	5,183	6,584	100.0	2,091.61	7,760
	yoyo	552	697	972	100.0	310.91	1,288
scpnrh1	std	2,631	3,454	4,416	100.0	1,225.83	5,199
	yoyo	247	330	491	100.0	99.37	728
scpnrh2	std	2,779	3,770	5,284	100.0	2,118.16	6,504
	yoyo	286	384	540	100.0	142.48	802
scpnrh3	std	2,606	3,445	4,629	100.0	1,563.01	5,656
	yoyo	287	396	545	100.0	154.62	798
scpnrh4	std	2,712	3,469	4,548	100.0	1,193.22	5,530
	yoyo	304	451	604	100.0	120.10	850
scpnrh5	std	2,482	3,484	4,739	100.0	1,298.52	5,724
	yoyo	258	372	527	100.0	100.22	761

Table 2: Set covering results under scenario B.

in the present paper, is not new in the Benders framework, as demonstrated by the seminal paper of Magnanti and Wong [15] and by the recent work of Naoum-Sawaya and Elhedhli [17]. As in the previous setting, our order of business was to solve by cutting planes the root-node LP relaxation of the MIP instance at hand. According to [16], this is a useful way for warm-starting the pool of cuts in a standard Benders' decomposition scheme.

Our testbed is made by instances of the so-called *multicommodity-flow network design problem* [2], where one has to allocate capacity to the arcs of a given network by ensuring that all commodities can simultaneously be routed from source to destination. This problem, as well as many other network design problems, is well suited for a Benders' approach because there is natural partition between first-stage integer variables (arc capacities to setup) and second-stage continuous variables (network flows)—see [10] for a recent survey on the subject.

problem	method	itr 90%	itr 95%	itr 99%	%Cl.Gap	totTime	totItr
scpclr11	std	3,470	5,473	-	98.8	807.25	10,000
	yoyo	249	316	757	100.0	706.28	4,087
scpclr12	std	7,723	-	-	93.6	946.48	10,000
	yoyo	73	171	740	100.0	1,160.06	3,885
scpclr13	std	7,136	-	-	93.8	2,399.71	10,000
	yoyo	123	187	1,064	100.0	8,872.70	4,804
scpnrg1	std	-	-	-	86.7	2,179.87	10,000
	yoyo	773	1,007	1,613	100.0	1,027.77	4,255
scpnrg2	std	-	-	-	88.7	1,913.55	10,000
	yoyo	749	993	1,511	100.0	766.45	4,138
scpnrg3	std	9,773	-	-	90.4	1,863.02	10,000
	yoyo	631	925	1,536	100.0	983.29	3,931
scpnrg4	std	-	-	-	86.6	1,986.56	10,000
	yoyo	755	1,022	1,576	100.0	941.53	4,007
scpnrg5	std	-	-	-	87.9	2,170.59	10,000
	yoyo	697	956	1,472	100.0	1,031.12	3,783
scpnrh1	std	7,767	-	-	94.4	2,133.01	10,000
	yoyo	349	489	886	100.0	496.20	2,633
scpnrh2	std	8,491	-	-	93.2	2,344.62	10,000
	yoyo	383	577	938	100.0	638.84	2,825
scpnrh3	std	7,361	-	-	95.0	2,874.08	10,000
	yoyo	398	592	957	100.0	636.59	2,905
scpnrh4	std	7,534	9,873	-	95.2	2,108.84	10,000
	yoyo	387	550	883	100.0	469.17	2,506
scpnrh5	std	7,900	-	-	94.2	2,896.72	10,000
	yoyo	314	464	797	100.0	456.18	2,449

Table 3: Set covering results under scenario C.

We generated two different types of random instances of the above network design problem, namely `grid` and `random`, according to the underlying network topology. We considered two different scenarios: in the `feas` case, routing costs were set to zero and only feasibility cuts were generated by the Benders separation procedure, while in the `opt` case each unit of flow had a cost of 1 on each arc, hence both feasibility and optimality Benders' cuts were generated. For these instances, the initial internal point was easily computed as $q = (1000, \ldots, 1000)$.

The outcome of our experiments is reported in Table 4, showing that yoyo search was about twice as fast as the standard method in producing the root-node LP bound, and required about 1/3 iterations (i.e., cuts)—both of them using exactly the same separation procedure.

4 Conclusions

We have investigated the search component of a cutting plane method. The standard search method commonly used for MIP problems is to cut an optimal vertex of the current LP relaxation. In that setting, however, the effectiveness of the resulting cutting plane method heavily depends on the availability of strong cuts, which is unfortunately not always the case when general MIPs are considered.

We have addressed the issue of designing an effective cutting plane scheme where the separation procedure is invoked to cut an "internal" (i.e., nonextreme) point of the current relaxation. The ellipsoid and analytic center methods are examples of such a scheme, that however are seldom used within a branch-and-cut MIP solution framework—though encouraging results have been reported very recently. Our approach is instead to keep the current MIP machinery (fast LP reoptimizations instead of interior point recomputations) and to modify the standard search method by exploiting the internal-point information to produce deeper cuts.

A possible implementation of the above idea is the yoyo-search scheme described in the present paper, where a sequence of internal points is generated with no computational overhead (except for the definition of the very first one, which is often an easy task). Computational results have been reported, showing that the new approach outperforms the standard one, mainly when shallow cuts are generated.

problem	itr std	itr yoyo	time std	time yoyo	#cuts std	#cuts yoyo
g_5_5_f_1	155	83	24.76	11.01	154	71
g_5_5_f_2	137	75	25.85	14.91	136	63
g_5_5_f_3	131	81	15.97	9.80	130	68
g_5_5_f_4	129	80	15.16	11.21	128	68
g_5_5_f_5	126	88	17.29	11.97	125	76
g_5_5_o_1	454	142	22.91	12.32	905	253
g_5_5_o_2	389	109	22.19	9.56	774	189
g_5_5_o_3	399	138	25.82	19.97	795	249
g_5_5_o_4	581	149	64.68	29.07	1158	271
g_5_5_o_5	270	83	5.15	4.87	538	145
g_5_6_f_1	220	108	216.99	152.43	219	97
g_5_6_f_2	205	113	220.59	132.33	204	101
g_5_6_f_3	73	60	9.49	5.42	72	50
g_5_6_f_4	137	106	106.53	105.55	136	93
g_5_6_f_5	109	100	62.34	66.99	108	88
g_5_6_o_1	731	136	129.91	60.64	1459	251
g_5_6_o_2	535	127	73.11	26.50	1064	226
g_5_6_o_3	463	104	56.29	22.29	922	184
g_5_6_o_4	473	122	81.44	44.01	943	217
g_5_6_o_5	197	144	37.80	50.39	393	259
r_20_5_f_1	148	89	28.04	16.36	147	75
r_20_5_f_2	120	62	15.02	6.16	119	49
r_20_5_f_3	188	103	44.45	25.03	187	88
r_20_5_f_4	106	69	14.75	7.37	105	56
r_20_5_f_5	852	133	452.02	62.76	851	119
r_20_5_o_1	523	85	12.76	3.98	1042	141
r_20_5_o_2	390	65	5.93	2.74	778	105
r_20_5_o_3	1056	124	66.74	11.04	2055	219
r_20_5_o_4	317	75	6.16	2.66	626	125
r_20_5_o_5	331	62	11.85	2.60	657	101
r_25_5_f_1	150	91	116.19	71.55	149	78
r_25_5_f_2	425	132	642.62	191.29	424	117
r_25_5_f_3	1418	231	3,591.81	644.01	1418	216
r_25_5_f_4	97	76	27.33	17.40	96	63
r_25_5_f_5	134	86	36.97	17.79	133	73
r_25_5_o_1	915	150	106.53	29.49	1823	273
r_25_5_o_2	780	114	62.78	15.64	1546	206
r_25_5_o_3	2041	241	441.66	111.01	4080	447
r_25_5_o_4	1728	196	469.09	77.34	3450	359
r_25_5_o_5	851	124	162.98	50.40	1675	222
geom.mean	314	105	50.74	22.86	441	129

Table 4: Benders' results.

References

[1] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. See http://miplib.zib.de.

[2] A. Atamturk and D. Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[3] D. S. Atkinson and P. M. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69:1–43, 1995.

[4] J. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. See http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

[5] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[6] R. G. Bland, D. Goldfarb, and M. J. Todd. The ellipsoid method: a survey. *Operations Research*, 29(6):1039–1091, 1981.

[7] S. Boyd and L. Vandenberghe. Localization and cutting-plane methods. available at http://www.stanford.edu/class/ee364b/notes/localization_methods_notes.pdf, 2007.

[8] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

[9] G. Cornuéjols and C. Lemaréchal. A convex analysis perspective on disjunctive cuts. *Mathematical Programming*, 106(3):567–586, 2006.

[10] A. Costa. A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32(6):1429–1450, 2005.

[11] J. Elzinga and T. J. Moore. A central cutting plane algorithm for the convex programming problem. *Mathematical Programming*, 8:134–145, 1975.

[12] J.-L. Goffin, Z.-Q. Luo, and Y. Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 6:638–652, 1996.

[13] J.-L. Goffin and J.-P. Vial. On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. *Mathematical Programming*, 60:81–92, 1993.

[14] J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[15] T. Magnanti and R. Wong. Accelerating Benders decomposition: algorithmic enhancement and model selection criteria. *Operations Research*, 29:464–484, 1981.

[16] D. McDaniel and M. Devine. A modified Benders' partitioning algorithm for Mixed Integer Programming. *Management Science*, 4:312–319, 1977.

[17] J. Naoum-Sawaya and S. Elhedhli. An interior-point branch-and-cut algorithm for mixed integer programs. Technical report, Department of Management Sciences, University of Waterloo, 2009.

[18] Y. Nesterov. Cutting plane algorithms from analytic centers: efficiency estimates. *Mathematical Programming*, 69(1):149–176, 1995.

[19] S. Tarasov, L. Khachiyan, and I. Erlikh. The method of inscribed ellipsoids. *Soviet Mathematics Doklady*, 37:226–230, 1988.

[20] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley, New York, 1997.