Submodular Minimization in the Context of Modern LP and MILP Methods and Solvers

Andrew Orso, Jon Lee, and Siqian Shen

Department of Industrial and Operations Engineering, University of Michigan
1205 Beal Ave., Ann Arbor, MI 48109-2117
{orso,jonxlee,siqian}@umich.edu

Abstract. We consider the application of mixed-integer linear programming (MILP) solvers to the minimization of submodular functions. We evaluate common large-scale linear-programming (LP) techniques (e.g., column generation, row generation, dual stabilization) for solving a LP reformulation of the submodular minimization (SM) problem. We present heuristics based on the LP framework and a MILP solver. We evaluated the performance of our methods on a test bed of min-cut and matroid-intersection problems formulated as SM problems.

Keywords: submodular minimization, Lovász extension, column generation, row generation, dual stabilization

1 Introduction

Let E be a finite (ground) set. A function $f : 2^E \to \mathbb{R}$ is *submodular* if:

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T) \quad \forall S,\ T \subseteq E.$$

The goal of submodular minimization (SM) is to choose $S \subseteq E$ such that $f(S)$ is minimized. SM has strong ties to problems in machine learning, graph theory, and matroid theory. For example, in graph theory, a cut function, evaluated as the sum of capacities of arcs that originate from a subset of nodes containing a given source s, but not containing a given sink t, to nodes in the complementary set, is well known to be submodular; thus, the minimum s-t cut problem can be recast as an SM problem (see [11], for example). Additionally, we can recast the max-cardinality matroid-intersection problem, as a SM problem (see [3]).

SM is well known to be solvable in polynomial time using the ellipsoid method (see [7]). But the practicality of such an algorithm is very limited. Iwata, Fleischer, and Fujishige (see [8]) and Schrijver (see [14]) developed the first "combinatorial" polynomial-time algorithms for SM, however, again the practical use of such algorithms is quite limited. An algorithm that has had the most success seeks the minimum-norm point of the "base polyhedron" (see [5]), but even that algorithm has been found to be slow and/or inaccurate (see [9]). So we regard the challenge of developing practically-efficient approaches for SM as open.

The aforementioned algorithms for SM take advantage of the Lovász extension of a submodular function. This function is an extension of a submodular

function, viewed as defined on the vertices of the unit hypercube $[0,1]^E$, to the entire hypercube, as a piecewise-linear convex function (see [11]). Using the Lovász extension, one can derive an equivalent linear-programing (LP) problem with a very large number of columns (see [13], for example). Solving this LP problem had been deemed highly impractical (see [1], for example). In what follows, we demonstrate that large-scale LP techniques can be employed with some success, giving evidence that the LP approach should not be abandoned. Finally, we consider the use of modern MILP solvers such as Gurobi for solving SM problems as 0-1 integer programs. We take advantage of heuristics and cutting-plane methods in these solvers to develop heuristics for approximate SM.

The remainder of this paper is organized as follows. In §2, we present an equivalent LP formulation, as well as column- and row-generation procedures for solving it. In §3, we present computational results of these methods to compare row and column generation, as well as their stabilized method variants. In §4, we present experimental results of using MILP solvers for solving the dual of our LP reformulation by utilizing solver cutting-plane methods. Finally, in §5, we make brief conclusions and give future research directions.

2 Large-Scale LP Representation

Without loss of generality, we assume that $f(\emptyset) = 0$. For $c \in \mathbb{R}^E$, we define $c(S) = \sum_{k \in S} c_k$. The *base polyhedron* of f is $B(f) := \{c \in \mathbb{R}^E \mid \forall S \subseteq E,\ c(S) \leq f(S),\ c(E) = f(E)\}$. Now, define $c_- \in \mathbb{R}^E$ by $(c_-)_k := \max\{c_k, 0\}$, for $k \in E$. The SM problem $\min\{f(S) : S \subseteq E\}$ can be recast as $\max\{c_-(E) : c \in B(f)\}$ (see [13], for example). This maximization problem can be formulated as an LP problem with a large number of variables as follows. Letting $c^1, \ldots, c^m$ be the extreme points of $B(f)$, and defining the matrix $C := [c^1, \ldots, c^m] \in \mathbb{R}^{E \times m}$, we have the equivalent LP problem

$$\begin{aligned} \textbf{SMP:}\ \min\ & \mathbf{1}^\top \beta && (1)\\ \text{s.t.}\ & Cx - \alpha + \beta = \mathbf{0}, && (2)\\ & \mathbf{1}^\top x = 1, && (3)\\ & x \in \mathbb{R}^m_+, \alpha \in \mathbb{R}^E_+, \beta \in \mathbb{R}^E_+, && (4) \end{aligned}$$

where $\mathbf{1}$ (resp. $\mathbf{0}$) is a vector of appropriate dimension with all entries equal to 1 (resp., 0). To construct a minimizer of f from a *basic dual optimum* of this LP problem, we consider the dual variables corresponding to constraints (2); these dual variables are binary and correspond to a minimizer of f (see [13], for example).

2.1 Column Generation

Column generation is a standard technique for handling LP formulations in which we have a manageable number of equations (in a standard-form LP problem) but a very large (but structured) set of variables. In our context, **SMP** has

only $|E|+1$ equations but m variables (which would typically be exponentially large, relative to $|E|$). Define **RSMP** to be a restricted version of **SMP**, in which C is replaced by $\tilde{C}$, having a subset of the columns of the full matrix C. We iteratively solve instances of **RSMP**, further incorporating variables that have negative reduced cost for the current **RSMP**, and possibly dropping variables that are nonbasic, after each iteration. Typically, a basis of **SMP** has $|E|+1$ variables, and so we maintain at least $|E|+1$ (but much fewer than m) variables in **RSMP**.

To determine which variable to add to the model at the end of each iteration, we solve a (typically) well-structured problem known as the pricing problem, which determines a variable, not already in the model, that has negative reduced cost. Our pricing problem is $\max\{u^\top c \,:\, c \in B(f)\}$, where $u \in \mathbb{R}^E$ is the vector of optimal dual variables corresponding to the constraints $\tilde{C}x - \alpha + \beta = \mathbf{0}$ in **RSMP**. This is simply the maximization of a linear function over the base polyhedron of f, which can be efficiently evaluated via the greedy algorithm (see [10], for example). Furthermore, we can take advantage of the structure of a particular function f to improve the greedy algorithm. For example, for min-cuts, the value of a cut during the greedy algorithm can be used in the subsequent greedy step, after adding one node, by subtracting the capacities of those arcs no longer in the cut and adding the capacities of the new arcs introduced.

The ease of solution of the pricing problem makes column generation a viable method for **SMP**, though a common pitfall of these methods is that they tend to "tail-off", where convergence begins to slow considerably as the solution nears the optimum. To combat such woes, we implemented a dual-stabilization technique, which has been shown to decrease these effects in practice (see [12], for example). In particular, the optimal dual solutions at each iteration tend to fluctuate considerably, rather than following a smooth trajectory. Dual stabilization seeks to localize the dual variables from iteration to iteration, hoping to moderate this behavior (see [12], for example). There exist numerous methods for stabilizing the dual problem (e.g., interior point stabilization (see [4]), bundle methods (see [2]), etc.). One such method is the box-step method: consider a dual optimal solution $(u^i, v^i) \in \mathbb{R}^E \times \mathbb{R}$ at iteration i of the column-generation procedure. We introduce new constraints to the dual, which is equivalent to adding columns to the primal, of the form $u_j^{i+1} \in [u_j^i - \epsilon, u_j^i + \epsilon]$ for all $j \in E$, where $\epsilon > 0$ denotes the user-defined width of the box. In the subsequent iteration, if the optimal dual solution u_j^{i+1} is at either the upper or lower bound of the box, then the box is recentered at that bound and the procedure continues.

2.2 Row Generation

Alternatively, we consider solving the dual of **SMP**

$$\textbf{SMD:} \quad \max \; v \tag{5}$$

$$\text{s.t.} \; C^\top u + \mathbf{1}v \leq \mathbf{0}, \tag{6}$$

$$\mathbf{0} \leq u \leq \mathbf{1}, \tag{7}$$

with variables $u \in \mathbb{R}^E$ and $v \in \mathbb{R}$ corresponding to equations (3) and (4) respectively. **SMD** typically has a very large number of constraints, as opposed to the large number of variables present in **SMP**. Of course, by the LP strong-duality theorem, the optimal values of **SMP** and **SMD** are the same. Moreover, in some sense **SMD** is more natural to consider since we are really after a dual basic optimum of **SMP**. For solving **SMD**, we consider a row-generation procedure, which is analogous to column generation for **SMP**. We will refer to the constraints generated as *greedy cuts*, as they are the cuts generated using the greedy algorithm at each iteration. Thus, we maintain a relaxed version **RSMD** of **SMD**, replacing C with $\tilde{C}$, as before. We can start with $\tilde{C}$ having no columns, and we employ the pricing problem from before to determine greedy cuts to be added at every iteration. We can apply dual stabilization again, by explicitly bounding the variables u in **RSMD**. The significance of solving **SMD**, as opposed to **SMP**, is not readily apparent, though computational tests on a number of problem instances demonstrates that row generation may perform better than column generation, depending on the structure of the problem instance.

3 Computational Results: LP

We tested our LP-based algorithms on classical min s-t cut and matroid-intersection problems.

For the min-cut problem, we produced RMFGEN networks (see [6]) as b copies of an $a \times a$ grid of vertices in which each vertex within a grid is connected to each of its neighbors, as well as a random vertex in the adjacent grid. The source is the lower-left vertex of the first grid, and the sink is the upper-right vertex of the last grid.

For the matroid-intersection problem, let $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ be matroids with independent sets $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively on the common ground set E. The matroid-intersection problem is the problem of finding the maximum-cardinality independent set that is in both $\mathcal{I}_1$ and $\mathcal{I}_2$. By a result of Edmonds (see [3]), this problem is equivalent to minimizing the submodular function

$$f(S) := r_1(S) + r_2(E \setminus S) \quad \forall S \subseteq E,$$

where r_i is the rank function of $\mathcal{M}_i$, $i = 1, 2$. For our test instances, we chose $\mathcal{M}_1$ to be the graphic matroid of a random connected graph on $p+1$ vertices and $2p$ edges, and $\mathcal{M}_2 = \mathcal{M}_1^*$ to be its dual. This choice of instances has relevance in determining whether a "bar-and-joint framework" in the plane is "minimally generically rigid" (see [10], for example).

We identify our test problems with the following key: **Type_y-z** where **Type** denotes the type of problem [MC = minimum cut, MI = matroid intersection], and **y-z** denotes the size, e.g., for MC, $y = a$, $z = b$, and for MI, $y = p$, $z = 2p$. The problem sizes we consider are given in Table 1.

All computations were performed using Gurobi 5.6.3 with a time limit of 12000 seconds on a Linux cluster with 4GB RAM per core, using four cores for

Table 1. Test Instances

Problem Sizes	
Minimum Cut	**Matroid Intersection**
4-32 (512 Nodes, 2032 Arcs)	150-300
8-8 (512 Nodes, 2240 Arcs)	175-350
7-13 (637 Nodes, 2772 Arcs)	190-380
9-9 (729 Nodes, 3200 Arcs)	200-400
5-32 (800 Nodes, 3335 Arcs)	

each MC instance, and one core for each MI instance. All results are given as an arithmetic average across 10 instances for each problem type.

3.1 Standard LP Formulations

We first tested the column- and row-generation procedures. Results are given in Table 2 and Table 3, for row and column generation, respectively. The $Time$ column reports the total time, in seconds, required to solve the corresponding problem to optimality, the $\#I$ column reports the total number of iterations required to solve to optimality, and the $Time_{j\%}$ and $\#I_{j\%}$ columns, where $j = 1$ or 5, report the amount of time and number of iterations required to reach a $j\%$ optimality gap, respectively.

Table 2. Results of Using Row Generation

Row Generation							
Problem Type	**Problem**	**Time**	**#I**	**Time$_{5\%}$**	**#I$_{5\%}$**	**Time$_{1\%}$**	**#I$_{1\%}$**
Minimum Cut	MC_8-8	493	826	282	480	360	602
	MC_9-9	2563	1218	1383	687	1769	851
	MC_4-32	494	803	263	432	364	617
	MC_5-32	3947	1330	1719	695	2688	998
	MC_7-13	1844	1075	938	616	1244	773
Matroid Intersection	MI_150-300	1044	284	110	28	830	223
	MI_175-350	2173	329	13	0	1568	235
	MI_190-380	9172	312	61	0	6937	211
	MI_200-400	3776	397	74	6	2502	259

Concerning the min-cut problem, we observe that while the number of iterations required to prove optimality is similar in the two methods, row generation requires considerably less time. Our tests indicate that the relaxed LP problems solved during each iteration of row generation require less time than the restricted LP problems solved during each iteration of column generation, resulting in an overall decrease in time. Similar results extend to the 5% and 1%

Table 3. Results of Using Column Generation

Column Generation							
Problem Type	**Problem**	**Time**	**#I**	**Time$_{5\%}$**	**#I$_{5\%}$**	**Time$_{1\%}$**	**#I$_{1\%}$**
Minimum Cut	MC_8-8	1017	819	407	480	621	602
	MC_9-9	4023	1186	1533	683	2303	847
	MC_4-32	950	816	331	434	575	624
	MC_5-32	9222	1333	2791	691	5238	993
	MC_7-13	2800	1054	1070	623	1739	797
Matroid Intersection	MI_150-300	1172	334	131	36	841	240
	MI_175-350	2407	370	13	0	1506	232
	MI_190-380	3103	377	16	0	1783	217
	MI_200-400	4273	434	62	4	2442	248

optimality gap, where row generation achieves these optimality gaps in fewer iterations and less time. Further, both methods require only about 50% of the total time in order to reach a 5% optimality gap and only about 75% of the total time in order to reach a 1% optimality gap. This quick, early convergence motivates the development of MILP heuristics in the next section.

For the matroid intersection, we see that row generation is only marginally better than column generation for the instances tested. While this result holds across many instances, instance MI_190-380 seems to be difficult for row generation. We will see that dual stabilization, introduced in the next section, helps to reduce this difficulty.

3.2 Dual-Stabilized Formulations

For the dual-stabilized variants of the row and column generation, we ran all problem instances with $\epsilon = 0.25$, where ϵ is the width of the box constraints. The results reported in Table 4 and Table 5 follow the same format as was described for the standard LP formulations.

Both methods derive some benefit from dual stabilization. In fact, both methods see a decrease in the number of iterations required to solve to optimality by approximately 15%. For the min-cut problem, row generation is still a clear front runner, though the improvement derived from stabilizing the dual problem seems to be consistent across both methods, in terms of time and iterations. For the matroid-intersection problem, stabilizing row generation yields a greater improvement in running time. This is in contrast to the standard LP methods, in which neither algorithm proved to be more effective.

4 Computational Results: MILP

In this section, we take advantage of the fact that extreme-point optima of **SMD** are integer and correspond to minima of f. We demonstrate that there

exists some number M of greedy cuts at which point **RSMD** can be solved to optimality with a pure integer-programming approach via branch and bound (B&B). We examine the use of this idea in solving **SMD**, as well as for heuristics.

Table 4. Results of Using Dual-Stabilized Row Generation

Dual-Stabilized Row Generation							
Problem Type	**Problem**	**Time**	**#I**	**Time$_{5\%}$**	**#I$_{5\%}$**	**Time$_{1\%}$**	**#I$_{1\%}$**
Minimum Cut	MC_8-8	432	738	225	466	308	571
	MC_9-9	1659	1124	824	689	1113	836
	MC_4-32	381	769	165	416	252	587
	MC_5-32	3234	1255	1251	686	2126	980
	MC_7-13	982	997	444	605	683	779
Matroid Intersection	MI_150-300	701	194	30	0	30	0
	MI_175-350	1260	209	50	0	50	0
	MI_190-380	1730	208	69	0	69	0
	MI_200-400	2427	250	81	0	81	0

Table 5. Results of Using Dual-Stabilized Column Generation

Dual-Stabilized Column Generation							
Problem Type	**Problem**	**Time**	**#I**	**Time$_{5\%}$**	**#I$_{5\%}$**	**Time$_{1\%}$**	**#I$_{1\%}$**
Minimum Cut	MC_8-8	1069	745	511	475	713	569
	MC_9-9	6118	1096	3049	702	3917	806
	MC_4-32	1010	752	393	416	677	598
	MC_5-32	8785	1177	3576	678	5861	930
	MC_7-13	3030	1005	1290	614	2027	776
Matroid Intersection	MI_150-300	832	240	29	0	37	2
	MI_175-350	1661	267	51	0	180	22
	MI_190-380	2219	275	66	0	66	0
	MI_200-400	3123	331	77	0	77	0

4.1 Submodular Minimization Using Integer Programming

The goal of this test was to explore the utility of modern integer-programming techniques for SM. We carried this out by a procedure we call "submodular minimization using integer programming" (SMIP). In the first phase of this method, we solve **SMD** to optimality using standard row generation and define K to be the number of greedy cuts necessary to verify optimality. By resolving **RSMD** with the first L greedy cuts generated, for some choice of L, the result is

a suboptimal solution of the LP problem **SMD**. But, we can pass this relaxation having L greedy cuts to an MILP solver and allow the MILP solver to run until completion, utilizing B&B, heuristics, cutting planes, etc. At completion, we can then run our greedy-cut-generation procedure again to determine if there exists any violated greedy cuts; if not, then the MILP solution is optimal to **SMD**, otherwise the MILP solver was unable to achieve a true optimum when provided with the L greedy cuts. We carried out such a scheme, continuing to decrement the number of cuts added to **RSMD**, until the MILP solution returned was not optimal to **SMD**, at which point we set M to the previous number of cuts for which the MILP solution was optimal for **SMD**. So, M/K is the proportion of greedy cuts actually needed to solve an instance to optimality in this MILP framework, which is typically very low in practice.

In the second phase, we initialized **RSMD** with the M sufficient constraints from the first phase and formulated **RSMD** as an integer program with variables $u \in \{0,1\}^E$. We solved the integer variant of **RSMD** as efficiently as possible with varying levels of cut aggressiveness under the MILPFocus Gurobi parameter that controls the focus on proving optimality of the given MILP.

We ran SMIP on MC_8-8, MC_4-32, and MC_7-13 as well as MI_150-300 and MI_175-350. We report the following results for these procedures: the number of iterations K required to solve instances to optimality using row generation, the number of iterations M sufficient to solve instances using a pure integer-programming method, the total amount of time, in seconds, for both phases, $Time$, the amount of time required to get the true optimal solution as an incumbent solution in the B&B search of the second phase, $Time_{\mathrm{I}}$, and the number of Gomory and MIR cuts generated.

The results reported in Tables 6 and 7 demonstrate that a pure integer-programming approach to solving these problems may be beneficial, especially in the case of matroid intersection, where the function evaluations during the greedy algorithm generally require a large number of computationally-expensive matrix-rank calculations. We also note the increase in the amount of time required to solve these problems as the aggressiveness of the cuts increases. The min-cut problem saw little benefit from this method as the average reduction in the number of iterations is not significant enough to warrant the long MILP solve time. In fact, compared to row generation, the times recorded are far worse. Conversely, for matroid intersection, the number of greedy cuts sufficient to solve to optimality in an MILP framework was approximately 60% of the total number required in the row-generation setting, while we required only a fraction of the time that would be required to generate the difference in cuts solving using integer-programming techniques. On the two matroid-intersection problems, the integer-programming approach performed similarly to dual-stabilized row generation.

4.2 Greedy-Cut Integer-Programming Method

In a similar vein, we can enhance SMIP by integrating greedy cuts throughout the B&B search in the second phase. We solved problems MC_4-32, MC_7-13,

Table 6. Results of Using SMIP on Minimum Cut

Min-Cut SMIP							
Size	**Cut Level**	**K**	**M**	**Time**	**Time$_I$**	**Gomory**	**MIR**
8-8	0	826	492	1192	15	0	0
	1	826	492	1193	10	6	6
	2	826	492	2220	27	12	2911
	3	826	492	3370	52	12	9780
4-32	0	803	548	1251	20	0	0
	1	803	548	903	23	5	0
	2	803	548	1462	61	13	302
	3	803	548	2060	100	12	1317
7-13	0	1070	761	1831	39	0	0
	1	1070	761	1827	47	4	0
	2	1070	761	2495	87	9	5575
	3	1070	761	2642	135	9	5575

Table 7. Results of Using SMIP on Matroid Intersection

Matroid-Intersection SMIP							
Size	**Cut Level**	**K**	**M**	**Time**	**Time$_I$**	**Gomory**	**MIR**
150-300	0	284	199	859	10	0	0
	1	284	199	835	4	11	0
	2	284	199	2354	722	29	0
	3	284	199	1710	721	27	0
175-350	0	329	251	1673	6	0	0
	1	329	251	2353	7	11	0
	2	329	251	3089	26	29	0
	3	329	251	3199	23	45	0

Table 8. Collection of Greedy Cut SMIP Data

SMIP Utilizing Greedy Cuts					
Problem Type	**Problem**	**K**	**M**	**Time$_{IP}$**	**Time$_{RG}$**
Minimum Cut	MC_4-32	803	507	539	493
	MC_7-13	1072	669	1664	1843
	MC_8-8	826	435	546	492
Matroid Int	MI_150-300	284	185	1077	1043
	MI_175-350	329	218	2076	2173
	MI_190-380	335	209	2426	9171

MC_8-8, MI_150-300, MI_175-350, and MI_190-380 and report the total number of cuts required to solve to optimality using row generation, K, the number of cuts sufficient to solve to optimality using the greedy cut MILP method, M, the average time, in seconds, to solve to optimality, $Time_{\mathrm{IP}}$, and the average time to solve the equivalent row-generation problem as a comparison point, $Time_{\mathrm{RG}}$.

The results in Table 8 for the min-cut problem demonstrate that this method is competitive with the standard row-generation method proposed previously. In fact, for a problem requiring a larger number of rows generated to solve to optimality, e.g., MC_7-13, the IP method performs better than the row-generation procedure. For matroid intersection, we see little improvement in terms of computation time over the row-generation method, and it is actually worse than SMIP without greedy cuts.

4.3 Heuristic Methods

We demonstrated not only the practicality of an MILP-based algorithm for solving SM problems, but also the relatively small amount of time in which the true optimal solution becomes an incumbent of the B&B search. We take advantage of this fact, in conjunction with local-search heuristics, to develop a fast method for getting good, approximate solutions to **SMP**, and equivalently **SMD**.

We use a simple local-search method in the heuristics that follow. Given a vector $u \in \{0,1\}^E$, we repeatedly change at most two components, so as to get the maximum decrease in f.

The first heuristic we develop can be employed in the context of row or column generation. We initialize **RSMD** (or **RSMP**) and proceed with row (resp., column) generation. At some point (say after a fixed number γ of iterations), we give up and switch to local search, starting that from a rounding of the primal (resp., dual) solution $u \in [0,1]^E$ to the nearest point in $\{0,1\}^E$. Results for the row-generation version are given in Table 9 with $\#Cuts$ being the number of greedy cuts generated before switching to local search, $Time$ being the total time in seconds, and $Ratio$ being the ratio of the difference between the optimal and heuristic value with the optimal value.

The second heuristic is identical to the first heuristic until local search is called. Before switching to local search, we run an MILP solver until some user-defined state is reached. A good option seems to be to run the MILP solver until three incumbent solutions of the B&B search have been found, at which point we start local-search from the best incumbent found. Computational results for SMIP demonstrate that on average, the true minimum of f is found as an MILP incumbent very quickly. Results for this method are given in Table 10, with columns indexed the same as in previous section, except $Time_{\mathrm{IP}}$, which is the time required to find the third incumbent of the B&B search.

From the results in Table 9, we see that the first heuristic competes with standard row-generation method for the min-cut problem, yielding shorter times and near optimal solutions in most cases. For the matroid-intersection problem, optimality is achieved in all cases, though the time benefit does not become significant until we consider problems of larger size. For the second heuristic,

Table 9. Results from heuristic method without MILP extension

Heuristic w/o MILP Extension				
Problem Type	**Problem**	**# Cuts**	**Time**	**Ratio**
Minimum Cut	MC_8-8	500 650	437 463	7.3 0
	MC_9-9	800 1000	1728 1964	0.02 0
	MC_4-32	600 700	525 463	0.37 0
	MC_7-13	600 700	972 1064	0.11 0.11
Matroid Intersection	MI_150-300	100 200	1243 1202	0 0
	MI_175-350	150 300	4144 2935	0 0
	MI_190-380	200 300	4106 4482	0 0

Table 10. Results from heuristic method utilizing MILP extension

Heuristic with MILP Extension					
Problem Type	**Problem**	**# Cuts**	**Time**	**Time$_{IP}$**	**Ratio**
Minimum Cut	MC_8-8	500 650	1515 1027	1110 453	0.77 3.49
	MC_9-9	800 1000	4095 3955	1708 667	1.55 3.62
	MC_4-32	600 700	1499 1059	925 327	1.97 1.96
	MC_7-13	600 700	2112 3742	1127 2487	1.55 2.24
Matroid Intersection	MI_150-300	100 200	371 667	4 5	0.02 0.05
	MI_175-350	150 300	935 1812	8 6	0.02 0.04
	MI_190-380	200 300	1588 2247	8 11	0.02 0.05

results in Table 10 indicate poor performance on the min-cut problem, with long solve times and larger ratios. Conversely, the time required to solve the matroid-intersection problem is cut down significantly and the ratio is very small.

5 Conclusion

We explored the applicability of modern LP/MILP methods for SM. For LP, we saw that row generation typically performs better than column generation, and we saw that MILP methods can help in an exact or heuristic context. We are currently expanding our tests and exploring how to incorporate side constraints.

Acknowledgement

This research was supported in part by Advanced Research Computing at the University of Michigan, Ann Arbor. J. Lee was partially supported by NSF grant CMMI–1160915 and ONR grant N00014-14-1-0315. S. Shen was partially supported by NSF grants CMMI–1433066 and CCF–1442495.

References

1. Bach, F.: Learning with submodular functions: A convex optimization perspective. Foundation and Trends in Machine Learning 6, 145–373 (2013)
2. Briant, O., Lemarechal, C., Meurdesoif, P., Michel, S., Perrot, N., Vanderbeck, F.: Comparison of bundle and classical column generation. Math. Prog. 113, 299–344 (2008)
3. Edmonds, J.: Submodular functions, matroids, and certain polyhedra. Combinatorial Structures and Their Applications pp. 69–87 (1970)
4. Elhedhli, S., Goffin, J.L.: The integration of an interior-point cutting plane method within a branch-and-price algorithm. Math. Prog. 100(2), 267–294 (2004)
5. Fujishige, S., Isotani, S.: A submodular function minimization algorithm based on the minimum-norm base. Pacific Journal of Optimization 7(1), 3–17 (2011)
6. Goldfarb, D., Grigoriadis, M.D.: A computational comparison of the dinic and network simplex methods for maximum flow. Ann. of OR 13(1), 81–123 (1988)
7. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2), 169–197 (1981)
8. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial strongly polynomial algorithm for minimizing submodular functions. JACM 48, 761–777 (2001)
9. Jegelka, S., Lin, H., Bilmes, J.: On fast approximate submodular minimization. In: Advances in Neural Information Processing Systems (NIPS), pp. 460–8 (2011)
10. Lee, J.: A First Course in Combinatorial Optimization. Cambr. Univ. Press (2004)
11. Lovász, L.: Submodular functions and convexity. In: Mathematical Programming The State of the Art, pp. 235–257. Springer (1983)
12. Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Operations Research 53(6), 1007–1023 (2005)
13. McCormick, S.T.: Submodular function minimization. Handbooks in Operations Research and Management Science 12, 321–391 (2005)
14. Schrijver, A.: A combinatorial algorithm minimizing submodular functions in strongly polynomial time. JCT, Ser. B 80(2), 346–355 (2000)