CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

On the Computational Efficiency of Subgradient Methods: A Case Study in Combinatorial Optimization

Antonio Frangioni
Enrico Gorgone
Bernard Gendron

August 2015

CIRRELT-2015-41

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

On the Computational Efficiency of Subgradient Methods: A Case Study in Combinatorial Optimization

Antonio Frangioni[1], Enrico Gorgone[2], Bernard Gendron[3,*]

[1] Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorve, 3 – 56127 Pisa, Italy

[2] Département d'Informatique, Université Libre de Bruxelles, Campus de la Plaine, C.P. 210/01, Boul. du Triomphe, 1050 Bruxelles

[3] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

Abstract. Subgradient methods (SM) have long been the preferred way to solve large-scale Nondifferentiable Optimization problems, such as those arising from the solution of Lagrangian duals of hard combinatorial optimization problems. Although other methods exist that show a significantly higher convergence rate in some circumstances, SM have certain unique advantages that may make them competitive under the right set of conditions. Besides, SM have significantly progressed in recent years, and new versions have been proposed with better theoretical and, in some applications, practical performances. We aim at computationally evaluating a large class of SM for the specific application of the solution of Lagrangian duals of integer programs, in order to assess if and how the main improvements suggested along the years have changed the computational landscape of these approaches. For this we first propose a unified scheme that covers many of the SM proposed in the literature, comprised some often overlooked, but computationally important, features like projection and dynamic generation of variables. This gives us a large class of SM, with several algorithmic parameters. We fine-tune and test them on two Lagrangian dual problems with significantly different characteristics, both arising from the same relevant and hard combinatorial optimization problem the - Fixed-Charge Multicommodity Capacitated Network Design problem - in order to assess the potential impact of the characteristics of the function to be minimized on the optimal algorithmic choices. Our results show that SM can be competitive with more sophisticated approaches when the tolerance required for solution is not particularly tight, which is what happens when Lagrangian techniques are used within exact or heuristic approaches for solving the original hard combinatorial optimization problem. However, for this to happen appropriate and nontrivial tuning has to be performed.

Keywords. Subgradient methods, computational analysis, Lagrangian relaxation, multicommodity network design.

Acknowledgements. The first author gratefully acknowledges the contribution of the Italian Ministry for University and Research under the PRIN 2012 Project 2012JXB3YF "Mixed-Integer Nonlinear Optimization: Approaches and Applications." The work of the second author has been supported by the Post-Doctoral Fellowship D.R. No 2718/201 (Regional Operative Program Calabria ESF 2007/2013-IV Axis Human Capital, Operative Objective M2, Action d.5) and the Interuniversity Attraction Poles Programme P7/36 "COMEX: combinatorial optimization metaheuristics & exact methods" of the Belgian Science Policy Office. The work of the third author has been supported by Natural Sciences and Engineering Research Council of Canada (NSERC) (Ca under grant 184122-09.

---

* Corresponding author: Bernard.Gendron@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2015

1. Introduction. The aim of this paper is to computationally evaluate a large family of Subgradient Methods (SM) for the (constrained) minimization of a convex nonsmooth function. We are in particular (although not exclusively) interested in problems of the form

$$(\Pi) \qquad f_* = \min \left\{ \ \sum_{k\in\mathcal{K}} f^k(\lambda) \ : \ \lambda \in \Lambda \ \right\} \tag{1.1}$$

where $\mathcal{K}$ is a given finite index set, $\Lambda \subseteq \mathbb{R}^n$ is closed, convex and "easy" in the sense that projection upon Λ is inexpensive, and $f^k : \mathbb{R}^n \to \mathbb{R}$ are proper convex functions that are finite-valued at least in an open neighborhood of Λ (or Λ itself if $\Lambda = \mathbb{R}^n$). The generalized gradient descent method, a.k.a. the *subgradient method*, is a generalization of the gradient method for smooth optimization introduced in the 60s [67] that solves (1.1) under very mild conditions. In particular, each of the functions f^k need only be known through a "black box" that, given $\lambda \in \Lambda$, computes the function value $f^k(\lambda)$ and a subgradient $g^k \in \partial f^k(\lambda)$. Then, after having aggregated all function values according to (1.1), and similarly for $\sum_{k\in\mathcal{K}} g^k = g \in \partial f(\lambda)$, the algorithm employs the simple recurrence formula

$$\hat{\lambda}_{i+1} = \lambda_i - \nu_i g_i \quad , \quad \lambda_{i+1} = \mathrm{P}_\Lambda(\hat{\lambda}_{i+1}) \, ,$$

where P denotes the orthogonal projection on Λ. Only very simple rules are required to the *stepsize* $\nu_i \in \mathbb{R}_+$ to ensure that the sequence $\{ f_i = f(\lambda_i) \}$ asymptotically solves (1.1), i.e., $\liminf_{i\to\infty} f_i = f_\infty = f_*$. Under mild additional assumptions, cluster points of $\{ \lambda_i \}$ also correspond to optimal solutions to (1.1).

The convergence rate of SM is rather bad: $\Theta(1/\epsilon^2)$ iterations are required in general to solve the problem up to absolute error ϵ, which means that attaining any more than a modest accuracy with these methods is, at least in theory, hopeless. Yet, SM has their advantages, and can be put to good use. For its minimalistic simplicity, which makes implementation not at all an issue, the method actually exhibits the *best* possible worst-case complexity for the minimization of a generic nondifferentiable function only known via a black box: in general, no algorithm can attain any better result than $\Theta(1/\epsilon^2)$ [55]. Besides, the complexity is independent of the size n of the problem; therefore, SM may be promising for very-large-scale problems where a high accuracy is not necessary, whereas a short running time is a primary concern. This happens to be often the case when f is the *Lagrangian function* of a hard optimization problem, say an Integer Program (IP). Indeed, in the following we will mainly refer to the *block-structured case*

$$\sup \left\{ \ \sum_{k\in\mathcal{K}} c^k u^k \ : \ \sum_{k\in\mathcal{K}} A^k u^k = b \ , \ u^k \in U^k \quad k \in \mathcal{K} \ \right\} \tag{1.2}$$

where one relaxes, in a Lagrangian fashion, the *complicating constraints* that link together blocks of variables that would otherwise be independent, i.e.,

$$f(\lambda) = \lambda b + \sum_{k\in\mathcal{K}} \left(\ f^k(\lambda) = \sup \left\{ \ (c^k - \lambda A^k) u^k \ : \ u^k \in U^k \ \right\} \right) \ . \tag{1.3}$$

We are mainly interested in the case that the sets U^k are "hard", e.g. encompassing integrality restrictions, so that (1.2) is a "hard" problem. Yet, the computation of f is obviously less hard than the solution of (1.2) simply because it decomposes into smaller—albeit possibly still hard—independent subproblems. There are also cases where f is simpler even if $|\mathcal{K}| = 1$ since U^1 has a specific structure that can be algorithmically exploited; sometimes, as in our applications, both effects actually

apply. Therefore, while we consider (1.3) as our prototype application, to simplify the notation we will write cu, $Au = b$ and U respectively for the objective function, linking constraints and feasible region in (1.2)/(1.3) when the sum-function structure is either not relevant, or better purposely ignored. We also remark that there is a slight (and intended) inconsistency between (1.3) and (1.1), in that the former actually is the sum of $|\mathcal{K}| + 1$ functions, counting the linear one λb; we will ignore this detail (e.g., assume $b = 0$) up until it becomes relevant.

It can be argued that Lagrangian relaxation of hard optimization problems, although clearly not the only application of SM, has been one of the main factors motivating the interest in this class of algorithms. Starting from the groundbreaking results of [38, 39] on the Traveling Salesman Problem, the definition of appropriate Lagrangian duals of IPs [34] has been a staple of integer optimization for over two decades; the latter article has been cited more than 1500 times. For nearly two decades, within integer programming circles, "Lagrangian relaxation" has invariably been a synonym of "solving a Lagrangian dual by a subgradient method." In fact, some of the developments trying to improve upon the (dreadful) computational issues in the original SM originates from the IP community. This is, for instance, the case of the *deflection techniques* introduced in [14] to "dampen" one of the main drawbacks of SM (in addition to these of the gradient method), that is, the fact that non-differentiability of f yields the "zig-zagging" behavior whereby the direction at one step is almost opposite to that of the previous step, so that two "reasonably long" steps combined in fact make an "unfeasibly short" one. This can be faced by slightly modifying the update rule as

$$\hat{\lambda}_{i+1} = \lambda_i - \nu_i d_i \tag{1.4}$$

where the direction d_i is obtained by some linear combination of g_i and the previous subgradient g_{i-1}. Actually, since then $d_{i-1} \neq g_{i-1}$, one uses the *previous direction* d_{i-1} to deflect the *current subgradient*. This is compounded in the constrained case by the fact that the direction $(-)g_i$ (or d_i) is chosen without taking the feasible set Λ into account, which also independently causes the zig-zagging phenomenon unless *conditional subgradient* techniques [48] are employed whereby g_i (d_i) is first projected on the tangent cone of Λ at λ_i. Again, IP has been the main motivation for their development: inequality constraints $Au \leq b$ in (1.2) give $\Lambda = \mathbb{R}^n_+$, where at least projection is very easy. It is somewhat surprising that the combination of these two techniques has not been considered until quite recently [21]. Also, stepsize rules have been developed specifically with a focus on integer optimization [6, 33].

However, the interest in SM has started to wane during the early 90s, for two different reasons. On the one hand, the success of polyhedral techniques—the seminal papers [62, 63], perhaps not coincidentally regarding the Traveling Salesman Problem, have been together cited more than 1200 times—has meant that Branch&Cut (B&C) approaches, usually based on standard Linear Programming techniques like the simplex method, have rapidly become the method of choice for the solution of IPs. On the other hand, methods for solving (1.1) with a less minimalistic stance on information accrual had been known for almost as long as SM [43], and appropriate variants have been developed over the years that have been gradually proven to be superior to SM in many circumstances. In particular, both Bundle methods [40, 50, 70] and center-based methods [22] (the latter often [35], but not always [61], based on interior-point techniques) *stabilize* the original cutting-plane method, most of the time resulting in better performances than both un-stabilized versions and SM [10, 13, 19, 31]. It is

fair to say, however, that the computational advantage of these methods upon SM is mostly seen "at the tail" of the convergence process, where SM convergence is rather slow, whereas Bundle methods can (if properly set) rapidly accrue "the right set of information to stop" [30, 31]. Conversely, at the beginning the behavior of the two classes of methods is not that much different, questioning the effectiveness (at least in that particular stage) of hoarding much more information than SM do [13]. This is especially so insomuch as retaining more information implies a higher cost per iteration due to the solution of the *master problem*; it can account for a significant, and even the largest, part of the total computational time [29, 30]. Although a number of techniques can be used to lessen the computational burden of the master problem, from solving it with specialized methods [24] to changing its form so that it can be more effectively solved by standard ones [8], none of these can make it so inexpensive as it is in SM. The master problem cost is particularly hurtful if one is interested in using parallel programming techniques in the block-structured case, for while the subproblems can naturally be solved in parallel, the master problem represents the sequential bottleneck that inherently limits the possible parallel scaling [15]. Finally, SM have been reported to be useful at the very least as "warm-starters" for other approaches, quickly providing them with reasonably good starting solutions that can then be efficiently turned into optimal ones [28, 29].

Furthermore, interest in SM picked up independently at least twice in the last 15 years. The first time was at the turn of the millennium [5, 6, 49], with the re-discovery of what should have been a well-known property [2], i.e., that SM can be endowed with ways to recover (approximately) feasible solutions to the *convexified relaxation* [51], that is, the primal equivalent of the Lagrangian dual. This puts them, at least in theory, on par with Bundle and center-based methods, that have always been well-known for being able to produce (good quality) primal solutions [23, 27] as a by-product of dual optimization. In turn, this puts Lagrangian techniques, irrespective of the specific algorithm used to solve the dual, on par (at least in theory) with linear programming based ones to serve as the computational workhorse behind Branch&Cut approaches [26]. More or less at the same time, *incremental* SM have been proposed [45, 54, 60] which for the first time allow to exploit the block-separable structure of (1.3) to potentially speed-up computations, something that—albeit with a very different set of trade-offs—Bundle methods were already well-known to be able to do [4, 10, 22, 30, 42].

A more robust upsurge of interest in SM is more recent, and is due to the development of *fast SM*. These are inspired by *fast gradient methods* [55], that have substantially better convergence rates than SM but require differentiability of f to work. The idea is that if f is nondifferentiable, but has a specific structure, then some *smoothing technique* can be used to "lend it just enough differentiability" for fast gradient methods to work. This approach has been pioneered in [57, 58] for functions of the form

$$f(\lambda) = \hat{f}(\lambda) + \max\{\, \langle \lambda, Au \rangle - \hat{\phi}(u) \;:\; u \in U \,\} \tag{1.5}$$

where $\hat{f}$ is convex, differentiable and its gradient is Lipschitz-continuous ($\hat{f} \in C^{1,1}$), U is a bounded closed convex set in a finite-dimensional real vector space, $\hat{\phi}$ is continuous and convex on U and A is a linear operator. So, f is nondifferentiable due to the second component; the core of the method consists in smoothing it, i.e., modifying the maximization problem as

$$\bar{f}_\mu(\lambda) = \hat{f}(\lambda) + \max\{\, \langle \lambda, Au \rangle - \hat{\phi}(u) - \mu d(\lambda) \;:\; u \in U \,\}$$

where the *prox-function* $d(u)$ is continuous and strongly convex on U (which recalls a *generalized Moreau-Yosida regularization* [25]). Provided that the maximization problem can still be solved efficiently, i.e., $\hat{\phi}$ and U are "simple", $\bar{f}_\mu$ is a smooth lower approximation of f, and the two minima can be related by a simple function of μ. Thus, one can apply a fast gradient to $\bar{f}_\mu$ and, appropriately managing μ, efficiently obtain an approximate solution to (1.1) by these means. This approach has been warmly welcomed in several applications that require the solution of large-scale convex optimization problems [53], mostly from fields like machine learning, data mining, inverse problems, and imaging (see for instance [1, 17]) that possess the required structure. In turn, the success in applications has stimulated a vibrant research stream that is producing new results. For instance, [11] presents a different smoothing technique so that the smoothed function can still be solved with a fast gradient method, whereas in [47] a different approach to the same smoothing as in [58] is proposed that uses the Frank-Wolfe method. The smoothing approach is revisited in [7], relaxing the assumption on the function and regarding both the Moreau-Yosida regularization [52] and Nesterov's smoothing [58] as a particular case.

Unfortunately, none of these methods are directly applicable in our context. We have written the above formulæ to underline the similarity with our (1.2)/(1.3) setting; however, we have $\hat{f} \equiv \lambda b$, i.e., "all the problem is in the maximization part," which is therefore "not easy." In other words, adding the smoothing term $\mu d(\cdot)$ is, in our applications, not really an option.

The nearest algorithm to fast SM that we are aware of and that can be applied in our setting it are primal-dual SM (PDSM) [56]. Interestingly, the recent universal fast gradient method [59] does not require any parameter, and it automatically switches from the fast gradient when f has the required properties to PDSM [56] when these are missing; for this reason, in this paper we concentrate on PDSM as the representatives of "modern" SM. Indeed, even the very recent [41], which combines in a unified framework PDSM with the Mirror-Descent method [55], provides a slight generalization that does not seem to dramatically enlarge the class of approaches that can be implemented.

The aim of this paper is to try to assess the possible role of best-of-breed among the current SM for the (approximate) solution of Lagrangian duals of IPs. In particular, we are interested in verifying whether the recent applicable developments in SM have (or not) significantly altered the computational significance of SM in this specific application. Our interest is motivated by the fact that, when used to provide lower bounds on (1.2), (1.1) has to be solved with an "intermediate" accuracy. In particular, the standard relative accuracy required to the solution of an IP is 0.01% (i.e., `1e-4`), and thus it would make no sense to solve (1.1) substantially more accurately than that. This value is, broadly speaking, not so coarse that a SM is clearly the best choice to attain it (as would, say, be `1e-2`), but as well not so fine as to be basically hopeless to attain with a SM (as would, say, be `1e-6`). This middle ground needs therefore to be explored computationally.

In order to do that, we first try to unify most of the known SM under a general scheme. We take inspiration from [21], where already a wide class of SM doing deflection and projection simultaneously is analyzed, and we add a number of other practically relevant issues such as several different forms of deflection and stepsize formulæ, and dynamic generation of Lagrangian variables. Thus, this paper should not be considered as just the computational part of [21], as many more issues are tackled. Also, we do not aim at providing contributions from a theoretical standpoint:

some of the variants that we have tested do not even have a rigorous convergence proof. The viewpoint here is that of the IP practitioner interested in just reliably getting a good enough bound on its problem quickly. We have instead developed an object-oriented `C++` code, which we plan to openly release in the future, that implements the proposed general scheme in a flexible way, so as to make it easy not only to obtain all the variants we describe, but also to add other ones without tampering with the skeleton of both the algorithm and its implementation.

The code has been tested on the solution of two different forms of Lagrangian dual of a particularly hard IP, the Fixed-Charge Multicommodity Capacitated Network Design (FC-MCND) problem [18]. The rationale for this choice is that while both relaxations exhibit the block-separable form (1.3), they differ—for the same FC-MCND instance—in the number of blocks, the number of Lagrangian multipliers, and whether or not the multipliers are constrained in sign. As we shall see, these characteristics have a significant impact on the optimal choice of the algorithmic parameters for SM, helping in better characterizing the strengths and weaknesses for each variant. However, the two relaxations ultimately compute the same bound, which allows for an interesting comparison between them as well as with other solution methods that attain the same bound, such as different algorithms to solve the same Lagrangian duals and even the use of general-purpose Linear Programming solvers.

The paper is organized as follows. In Section 2, we discuss the main characteristics of the applicable subgradient approaches presented in the literature, and we propose a unified algorithmic scheme that encompasses all of them. Section 3 is dedicated to our extensive numerical experiments: first, we describe the target FC-MCND problem and its two different Lagrangian relaxations, then we describe the experimental setup and the tuning phase, and finally, we present the results of the best SM variants, briefly comparing them with other approaches. These results, and the learned lessons, are summarized in the final Section 4. The Appendix contains the details of all SM we have used, as well as a specific recount of their algorithmic parameters and of the tuning we have performed on them.

2. A general subgradient scheme. In this section we discuss the basic building blocks of SM, and we describe a general scheme that allows to obtain most of the variants proposed so far in the literature.

2.1. Building blocks of subgradient methods. In order to construct a SM, a number of elements have to be chosen. While, in general, each decision interacts with all the others, most of them can be, at least on the outset, described independently. This characteristic is exploited in our `C++` implementation, where a base class `Subgradient` (in turn deriving from an abstract base class `NDOSolver`, setting a generic interface for nonsmooth optimization), other than having a number of algorithmic parameters in itself, uses two external classes, `Stepsize` and `Deflection`, to which some of the main algorithmic decisions are demanded. By implementing new derived classes from these two one can easily add new algorithmic variants to the scheme. We now briefly discuss the basic "building blocks" for SM, with the fine details provided in the Appendix.

2.1.1. Stepsize rules. Clearly, a crucial aspect of any SM is the selection of the *stepsize* ν_i. One of the surprising properties of these algorithms is that the stepsize can be in fact chosen without any knowledge, either a-priori or a-posteriori, of the specific function to be minimized; indeed, any choice of the stepsize satisfying the

so-called *diminishing/square summable* condition

$$\sum_{i=1}^{\infty} \nu_i = \infty \quad , \quad \sum_{i=1}^{\infty} \nu_i^2 < \infty \ ,$$

of which $\nu_i = 1/i$ is the prototypical example, can lead to a convergent algorithm. While this emphasizes the extreme robustness of this class of approaches, that basically "converge with the least possible help," these stepsizes are most often inefficient in practice. The archetype of efficient Stepsize Rules (SR) is due to Polyak [64], and simply reads $\nu_i = \beta_i(f_i - f_*)/\|g_i\|^2$, where $\beta_i \in (0, 2)$ is arbitrary. This, however, needs to be revised, both because the direction d_i taken by the algorithm is not, in general, (the one of) g_i (see section 2.1.2), and because the optimal value f_* is not known in general. Indeed, in our application computing it, or at least a tight upper bound, is precisely the aim of the exercise. These facts lead to the Polyak-type *target value SR* of the form

$$\nu_i = \beta_i(f_i - f_i^{lev})/\|d_i\|^2 \tag{2.1}$$

where f_i^{lev} is some current approximation of f_*. Over the years, several SR of this type have been proposed; see, e.g., [5, 6, 12, 20, 44, 66]. Except in specific cases that will be discussed separately, all of our SR will have this form.

One of the crucial points in (2.1) is, clearly, how f_i^{lev} is determined. In general, during the minimization of f one only has the *upper bound* $f_i^{rec} = \min\{ f_l : l = 1, \ldots, i \}$ (the *record value*) on f_*, but no lower bound. Because one wants $\nu_i \geq 0$, f_i^{lev} needs in general be smaller than f_i, and it completely makes sense to choose it smaller than f_i^{rec}; this leads to *target following* approaches where $f_i^{lev} = f_i^{rec} - \delta_i$ for some dynamically revised $\delta_i > 0$. Technically, these approaches are divided into *vanishing* and *nonvanishing* ones according to the fact that $\delta_i \searrow 0$ as $i \to \infty$ or not [21, 46, 65]. However, our application has a specific benefit in this respect because one often has a *lower bound* on f_* available: this is provided by the cost $c\bar{u}$ of some (the best found so far) feasible solution of (1.2), i.e., such that $\bar{u} \in U$ and $A\bar{u} = b$, as weak duality ensures that $c\bar{u} \leq f_*$. In practice, $\bar{u}$ may not always be available: either initially because no feasible solution has been found yet, or because (1.2) is actually empty. Furthermore, during a B&C approach one may be solving a restriction of the original (1.2), and therefore $c\bar{u} > f_*$ may happen. However, as soon as one obtains $f_i^{rec} \leq c\bar{u}(1 + \varepsilon)$, where ε is the required relative accuracy for the solution of (1.2), then the algorithm can be (and is, in practice) stopped right away. Hence, in our tests we will assume that a lower bound $\underline{f} \leq f_*$ is available, which provides a workable f_i^{lev} without a need for target following techniques to be used. This allowed us to reduce the set of SR to be tested to only the following three:

1. The `Polyak` rule [64], whereby β_i and f_i^{lev} are constant (they do not depend on i).
2. The `ColorTV` rule as implemented in the original Volume algorithm [5]. This is based on classifying the iterations based on the obtained improvement $\Delta f_i = f_{i-1} - f_i$, where *green* iterations correspond to a large improvement and *red* ones (roughly speaking) to a negative one. Actually, an iteration is only declared green if also the scalar product $\langle g_i, d_{i-1} \rangle$ indicates that the previous direction still is of descent at λ_i, being marked as *yellow* otherwise. The stepsize parameter β_i is then managed looking at consecutive iterations of the same color, being significantly increased after a pre-set number of green ones, slightly increased after a set of yellow ones, and markedly decreased after a set of red ones. Furthermore, f_i^{lev} is adjusted whenever $f_i < f_i^{lev}$, unquestionably proving that it is not a valid lower bound.

3. The `FumeroTV` rule introduced in [33], which changes both β_i and f_i^{lev}. At the beginning, f_i^{lev} is set to the available lower bound $\underline{f}$, but it is exponentially revised towards f_i^{rec} as the algorithm proceeds. The rule actually distinguishes two phases. In the first one, $\underline{f}$ "significantly contributes" to f_i^{lev}, β_i is decremented after a pre-set number of consecutive not improving iterations and never incremented. In the second phase, where the contribution of $\underline{f}$ to f_i^{lev} remains unchanged, β_i is decremented in the same way but it is also incremented after a pre-set number of consecutive improving iterations.

The exact details and parameters of these stepsize rules are described in the Appendix. It would be relatively straightforward to test other approaches, such as the standard target following ones [21, 46, 65], as our `C++` implementation does not postulate any specific property of the SR. In fact, other than the above three Polyak-type rules, we have also tested an entirely different RS, closely tied with the deflection rule of the PDSM, as discussed next.

2.1.2. Deflection. The non-differentiability of f is one of the causes of the zig-zagging phenomenon, whereby it may occur that $g_i \approx -g_{i-1}$; thus, two "reasonably long" steps ν_i and ν_{i-1} result in an overall "very short" total movement after the two iterations, slowing down convergence. To overcome this problem, it was proposed early on [14] to *deflect* the subgradient g_i, using the modified update rule (1.4) to compute the next iterate. A fairly (although not entirely) general version of the formula then reads

$$d_i = \alpha_i g_i + (1 - \alpha_i) d_{i-1} \tag{2.2}$$

for the *deflection parameter* $\alpha_i \in [0, 1]$. The use of a convex combination is crucial in the analysis, because it ensures that d_i is always an approximate (conditional, see section 2.1.3) subgradient of f, as recalled in section 2.2. Furthermore, this allows to produce (hopefully, asymptotically feasible) primal solutions $u \in conv(U)$ that are useful, e.g., for the active-set strategy, as discussed in section 2.1.5. We have therefore elected to restrict ourselves to only this class of Deflection Rules (DR). Note that the obtained direction d_i is to be scaled by the stepsize ν_i, so two multipliers not summing to one can always be scaled (up or down) as to do so, with the scaling factor then accounted by the stepsize. Still, there are several possible ways to choose the deflection parameter. In particular, to avoid the zig-zagging phenomenon one may chose it in such a way that $\langle d_i, d_{i-1} \rangle \geq 0$. Indeed, the original rule [14] was to compute α_i as the maximum value guaranteeing that property, which means $\alpha_i = 1$, i.e., no deflection at all, if $\langle g_i, d_{i-1} \rangle \geq 0$ already. For our experiments, we have considered the following three DR:

1. The `STSubgrad` rule corresponding to the original SM [64], where we do not have any deflection, i.e., $\alpha_i = 1$.
2. The `Volume` rule where α_i is chosen as the (safeguarded) optimal solution of the one-dimensional quadratic problem that, roughly speaking, seeks to find the least-norm convex combination of g_i and d_{i-1}. This is used in the Volume algorithm [6], and basically makes it behave mostly like a *poorman's Bundle method* [5], i.e., one where systematically only two subgradients are kept.
3. The `Primal-Dual` rule proposed in [56] for PDSM. This is a sophisticated variant in which α_i and ν_i (see section 2.1.1) are chosen *simultaneously* in order to obtain optimal worst-case estimates on the SM convergence rate. Following [56], we have implemented both the *simple averages* and the *weighted averages* variants.

Again, full details of the three approaches are given in the Appendix. We remark here that the fact that PDSM set both the stepsize and the deflection parameter together is easily accounted for in our `C++` code by just having the corresponding `Primal-Dual` class to derive from *both* the abstract classes `Stepsize` and `Deflection`. This shows that while the general scheme depicts the two aspects as independent, there is no problem when they actually have to be synchronized.

2.1.3. Projection. A second, independent source of zig-zagging is due to the fact that, in the standard SM, the direction d_i does not take into account the feasible set Λ. When λ_i is on the boundary of Λ and g_i is almost orthogonal to its frontier, λ_{i+1} may remain very near to λ_i even for a large stepsize ν_i, which can unduly slow convergence. To overcome this problem one can project $-g_i$ (when there is no deflection, $-d_i$ in general) over the *tangent cone* T_i of Λ at λ_i. This corresponds to considering the *essential objective* $f_\Lambda(\lambda) = f(\lambda) + \imath_\Lambda(\lambda)$, where $\imath_\Lambda$ is the indicator function of Λ. It is well-known that the *normal cone* N_i to Λ at λ_i, which is the polar of T_i, is $\partial\imath_\Lambda(\lambda_i)$. Projecting on T_i is then effectively choosing some $w_i \in \partial\imath_\Lambda(\lambda_i)$ in order to use $g_i + w_i$, a subgradient to the essential objective f_Λ, instead of just g_i, to define the direction. While there does not seem to be a reason not to do this, at least if Λ is easy to project upon, things are more complex if deflection also is performed. In fact, there are then 8 possible deflection schemes, corresponding to all possible combinations to projecting g_{i-1}, d_{i-1} and d_i. If one requires a formal proof of convergence (which we prefer but do not necessarily require), some specific safeguards are needed depending on the specific stepsize formula employed. According to [21], there are two different main schemes to attain convergence: the first is the *stepsize-restricted* (or deflection-first) one, whereby we condition ν_i on the basis of α_i, the second is the *deflection-restricted* (or stepsize-first) one, whereby we, conversely, condition α_i on the basis of ν_i. The first is limited to stepsize rules of the form (2.1), and requires the satisfaction of the *safe rule*

$$\beta_i \leq \alpha_i \ , \tag{2.3}$$

(which implies $\beta_i \leq 1$) ensuring that a step over a direction that is very far from $-g_i$ cannot be too large. In deflection-restricted approaches, one can rather choose ν_i arbitrarily (e.g. with a diminishing/square summable SR) provided that α_i is kept "large enough" by

$$(\nu_i \|d_{i-1}\|^2)(f_i - f_i^{lev} + \nu_i \|d_{i-1}\|^2) \leq \alpha_i \ . \tag{2.4}$$

We also mention that if projection were too expensive, which is not the case in our applications, one could substitute it with partial projections-like on working onto the individual constraints sets and not onto the entire feasible region, as advocated in [16]. The introduction of this approximation would not change much the algorithmic scheme presented in this paper. Besides, complex dual constraints are comparatively rare in our preferred application.

2.1.4. Incremental approaches. When f is a sum function and $|\mathcal{K}|$ (the number of components) is very large, the total cost of the oracle for computing $f(\lambda_i)$ may be large even if each of the components is, taken individually, quite inexpensive. This is, in part, the case in our specific application. Motivated by some developments in training approaches for machine learning, *incremental* subgradient variants have been developed where one operates on a single component f^k instead of the entire function f. This means that while computing the direction, one replaces the "full"

subgradient g_i by that g_i^k of one component $k \in \mathcal{K}$. Ideally, a sequence of *incremental* (or inner) iterations performed along single-component subgradients could be roughly as effective for nearing λ_i to an optimal solution as a sequence of *full* (or normal, or outer) iterations, while the function evaluation cost is reduced by a factor of $1/|\mathcal{K}|$ [9, 60]. However, to guarantee convergence one needs to regularly compute the whole function f, so not all the iterates can be incremental. Besides, due to the risk that a step along one "rogue" component may move λ_i away from the optimum, the stepsize of incremental iterations need to be reduced with respect to that of full ones. Compounded with the fact that the cost of computing the next iterate, given g_i^k, is not substantially reduced in an incremental iteration, it is not given that the incremental approach is competitive in all applications.

2.1.5. Active set. When n—the number of variables in λ, i.e., the number of complicating constraints in (1.2)—is particularly large, it makes sense to use an *active set* strategy whereby only a subset of them is considered (given a nonzero value) at each iteration. This is particularly true for the case where the complicating constraints in (1.2) are inequalities, and hence $\Lambda = \mathbb{R}^n_+$, because one can expect that only a fraction of them will actually be binding at optimality. Indeed, if the set of complicating constraints is exponentially large (but an efficient separator, be it exact or heuristic, is available), there is no other chance but to proceed in this way; this is known as "Relax-and-Cut" in the literature [37]. Even if the constraints are polynomially many, active-set techniques can still be very useful [27, 30, 31]. The relevant technical issue is what primal vector $\bar{u}_i$ is used to perform separation, i.e., to identify violated constraints to be added to the active set. The obvious choice in the context of Lagrangian relaxation is the optimal solution u_i of the Lagrangian problem corresponding to the current iterate λ_i. However, a more sound technical choice is to use the *convexified solution* that can be generated at each iteration [2, 5, 6, 36, 49] and that, under appropriate conditions, converges to the optimal solution of (1.2) (if it is a convex problem, of its convexified relaxation otherwise). This is computed by taking convex combinations of the primal solutions generated during the algorithm; following (2.2), it is simply obtained as $\bar{u}_i = \alpha_i u_i + (1-\alpha_i)\bar{u}_{i-1}$. From the theoretical viewpoint, the active-set technique poses little convergence issues: if the active set is monotonically increasing, every convergent algorithm will remain convergent if the active set strategy is used (eventually, all variables will be in the active set and then "true convergence" will begin). Careful removal of variables from the active set is also possible; however, in practice the approach works without much issues.

2.1.6. Summary. All these aspects give rise to a rather large set of possible combinations, many of which have algorithmic parameters that have to be tuned for optimal performances. Not all of these combinations have reliable proofs of convergence, although several do. For instance, combining projection and deflection was studied in [21], but the incremental approach was not addressed there. The combination of incremental and deflection was studied in [69]. Yet, as already remarked, our interest was primarily in the computational behavior of the approaches; barring dramatic mis-settings of the algorithmic parameters, all the ones we have tested showed at least some degree of convergence, as our results will show, even in the absence of a formal convergence theory. This conforms with the well-known fact that SM are remarkably robust approaches, even if, possibly, at the cost of convergence speed.

We also remark that we have not (and, realistically, could not have) tested all possible variants of SM. Among the techniques that have been left out of the experiments are space-dilation methods [45, §7.2], other SR like variants of the Polyak stepsize us-

ing slightly modified terms at the denominator [45, (7.11)] or Ermoliev-like stepsizes [45, (7.6)–(7.9)], the *heavy ball SM* [68] popular within backpropagation approaches to train neural networks, and others. Yet, the structure of our `C++` code would allow to easily incorporate most of these variants.

2.2. A generic subgradient scheme. We now present a generic scheme of SM, in order to be able to discuss all its components (and their nontrivial interactions) individually.

0. Select $\bar\lambda_0 \in \Lambda$; $\lambda_1 \leftarrow \bar\lambda_0$, $d_0 \leftarrow 0$, $i \leftarrow 0$ and go to step 4;
1. Possibly, $d_{i-1} \leftarrow \mathrm{P}_{T_i}(\, d_{i-1}\,)$;
   if(`StepRes`) then $\alpha_i =$ `Deflection()`; `ComputeD()`; $\nu_i =$ `Stepsize(`α_i`)`;
   else $\nu_i =$ `Stepsize()`; $\alpha_i =$ `Deflection(`ν_i`)`; `ComputeD()`;
2. If some stopping test is satisfied, exit;
3. $\lambda_{i+1} \leftarrow \mathrm{P}_\Lambda(\, \bar\lambda_i - \nu_i d_i\,)$;
4. if(`OutItr`) then evaluate $f_{i+1} = f(\lambda_{i+1})$ and $g_{i+1} \in \partial f(\lambda_{i+1})$;
   else select k, evaluate $f^k(\lambda_{i+1})$ and $g_{i+1} \in \partial f^k(\lambda_{i+1})$;
5. Possibly, $g_{i+1} \leftarrow \mathrm{P}_{T_i}(\, g_{i+1}\,)$. Select $\bar\lambda_{i+1}$, set $\bar f_{i+1}$ accordingly;
6 $i \leftarrow i+1$ and go to step 1.

In the following we discuss the common features of all the variants we have implemented under this scheme and we give some general remarks.

- In the scheme, the new iterate is generated at Step 3 starting from the *stability center* $\bar\lambda_i$, which is updated at Step 5. In the classical SM the updating is automatic: $\bar\lambda_{i+1} = \lambda_{i+1}$. In the parlance of Bundle methods, this is called a *Serious Step* (SS), as opposed to *Null Steps* (NS) where instead the stability center is not moved: $\bar\lambda_{i+1} = \bar\lambda_i$. Intuitively, $\bar\lambda_i$ could be regarded as the best iterate found so far: moving to a different solution only makes sense insomuch as this actually leads to an improvement ($\Delta f_i > 0$, see section 2.1.1), possibly a *significant* one, whereas a NS may be preferable if the objective function value worsens (or grows too little). This is in fact done by some subgradient schemes, such as the `Volume` algorithm [6, 66] or its variants [5]. Basically, this makes SM equivalent to *poorman's Bundle algorithms*, where sequences of NSs aim at making d_i a better (descent) direction for $\bar\lambda_i$. Interestingly, PDSM provide an entirely different rationale for using a stability center (without ever changing it, see the Appendix); our general scheme allows to cover all these cases. In our software framework, the choice for $\bar\lambda_i$ is limited between the two options provided by SS and NS, as all SM variants we are aware of only employ these (note that Bundle methods instead exist that can make different choices [3]). As the selection of the next stability center is mostly tied to the update of d_i, we demand this choice to the `Deflection()` object. All this requires some quite obvious changes in some of the standard formulæ. In particular, the function improvement at each iteration (upon which the SS versus NS decision may be made) has to be defined as $\Delta f_i = \bar f_{i-1} - f_i$, and one has to use $\bar f_i$ instead of f_i in (2.1).
- In the scheme, `ComputeD()` computes the current direction using the extension of (2.2) reading

$$d_i = \alpha_i \bar g_i + (1-\alpha_i)\bar d_{i-1} \ ,$$

  where $\bar g_i$ and $\bar d_{i-1}$ are either g_i and d_{i-1} or their projection over the *tangent cone* T_i of Λ at $\bar\lambda_i$. Note that, reasonably, the tangent cone *at the stability*

center is used, which may be different from the last iterate λ_i, as discussed above. Furthermore, possibly $d_i \leftarrow \mathrm{P}_{T_i}(d_i)$, which is also performed within `ComputeD()`; hence, all 8 possible deflection schemes allowed by the theory are permitted. However, projecting all three times is useless, and therefore is avoided, because T_i is convex and we perform convex combinations, so if both g_{i-1} and d_{i-1} are projected, then $d_i \in T_i$ already. Furthermore, not all of these in principle result in a provably convergent algorithm (see [21, Lemma 2.4]), but this is not an issue in our computational setting: the un-attractive versions will be weed out by their practical efficiency.

- The behavior of the algorithm is somewhat different if ν_i is computed after d_i (stepsize-restricted) or vice-versa (deflection-restricted), as controlled by the algorithmic parameter `StepRes` (which may ideally change throughout the algorithm, but that we always keep fixed). Because computing d_i requires the choice of α_i, `ComputeD()` in the scheme always comes after `Deflection()`. However, in the deflection-restricted approach, the safe rule (2.3) requires ν_i in order to choose α_i, and consequently `Stepsize()` has also to be called before `ComputeD()`. Note that, in this case, when using Polyak-type formulæ, one would require $\|d_i\|$ before having computed d_i; our choice has been to replace it by $\|d_{i-1}\|$. The stepsize-restricted case is more natural for Polyak-type stepsizes, in that d_i is fully computed before ν_i. In PDSM, ν_i and α_i are chosen simultaneously, and therefore `StepRes` has no effect.
- Again concerning projection, in order to achieve theoretical convergence, the safe rules (2.3) and (2.4) should be used in the stepsize-restricted case and in the deflection-restricted one, respectively. Again, since we are mainly interested in practical performances we allow to switch off the safe rules in our code, in order to test whether the theoretical convergence guarantee they offer actually translates in better performances.
- We update the active set (if the approach is used) using the primal solution $\bar{u}_i$ (see section 2.1.5), which depends on the choice of α_i. Hence, the active set can only be updated after that `Deflection()` has been called. However, if the active set changes, then the vectors d_{i-1} and g_i need to be updated to take into account the new components, which in turn may change α_i. Hence, after an active set update, we compute again the deflection parameter α_i, and in the deflection-restricted scheme also the stepsize; the process is repeated until the active set remains unchanged.
- Again regarding the active set, if any projection on the tangent cone is performed, then this will have to be done each time new variables are added. Fortunately, in all of our (constrained) applications $\Lambda = \mathbb{R}^n_+$, so that the projection can be computed component-wise: only the new components need be dealt with. In general, however, adding new components may require re-projecting the entire vector; in turn, this would require keeping unprojected versions of d_{i-1} and g_i, roughly doubling memory requirements of the method (not that this is usually a bottleneck, at least in our applications). In order to avoid this, we support the option that only the new parts of these vectors are projected, thus ignoring the problem, even if this does not result in the direction that the theory would dictate. Of course, in this case, the final d_i need to be projected even if both d_{i-1} and g_i have, because their projection might have been inaccurate. However, all this is irrelevant for our experiments where component-wise projection is exact.

- The *linearization error* of g_i at $\bar{\lambda}_i$ is the (nonnegative) number

$$\sigma_i = \sigma_i(\bar{\lambda}_i) = \bar{f}_i - [f_i + \langle g_i, \bar{\lambda}_i - \lambda_i \rangle] = \sigma_i(\bar{\lambda}_{i-1}) - \Delta\bar{f}_i - \langle g_i, \bar{\lambda}_i - \bar{\lambda}_{i-1} \rangle \quad , \quad (2.5)$$

  where $\Delta\bar{f}_i = \bar{f}_{i-1} - \bar{f}_i$. Note that $\Delta\bar{f}_i$ differs from Δf_i when a NS occurred at iteration $i-1$, i.e., $\bar{\lambda}_i = \bar{\lambda}_{i-1} \Longrightarrow \Delta\bar{f}_i = 0$. Convexity of f ensures that $\sigma_i \geq 0$ and $g_i \in \partial_{\sigma_i} f(\bar{\lambda}_i)$. Furthermore, σ_i can be easily kept updated when $\bar{\lambda}_i$ changes by the *information transport property* (2.5). The linearization error may play a role at different points in the algorithm, such as some of the deflection rules (see the Appendix) and the stopping tests (see next paragraph). However, when projection is used, one rather wants to compute the linearization error of the *projected* $\bar{g}_i$, which is a subgradient of the essential objective $f + \imath_\Lambda$. This is why the projection of g_i is not performed at Step 1, but it occurs before updating $\bar{\lambda}_i$ at Step 5: so that, in case of a SS, the linearization error of $\bar{g}_i$ is computed (through (2.5) where $g_i = \bar{g}_i$ because projection has been performed at the beginning of Step 5). A downside of this choice is that if the current point $\bar{\lambda}_i$ changes at Step 5 (a SS), then at the beginning of the next iteration g_i may have to be projected again; however, projections (if at all required) are inexpensive in our applications. Furthermore, if a NS is performed then this is not needed.
- An advantage of (2.2), which underlines all the analysis in [21], is that we can similarly compute and keep updated the linearization error of d_i. That is, knowing that $d_{i-1} \in \partial_{\epsilon_{i-1}}(\bar{\lambda}_i)$, it is immediate to realize that $d_i \in \partial_{\epsilon_i} f(\bar{\lambda}_i)$ with $\epsilon_i = \epsilon_i(\bar{\lambda}_i) = \alpha_i\sigma_i(\bar{\lambda}_i) + (1-\alpha_i)\epsilon_{i-1}(\bar{\lambda}_i)$. Since, as usual, linearization errors can be cheaply updated after a SS with the information transport property, which in this case reads

$$\epsilon_i(\bar{\lambda}_{i+1}) = \epsilon_i(\bar{\lambda}_i) - \Delta\bar{f}_{i+1} - \langle d_i, \bar{\lambda}_{i+1} - \bar{\lambda}_i \rangle \quad ,$$

  we have that d_i is a $\epsilon_i(\bar{\lambda}_{i+1})$-subgradient of f at $\bar{\lambda}_{i+1}$. This means, however, that the same issue about projection arises here also.
- In SM, it is possible to use the inverse of $\|g_i\|$ in (2.1) because as soon as $\|g_i\| = 0$, one has proven the optimality of λ_i. Since $g_i \in \partial_{\sigma_i} f(\bar{\lambda}_i)$, this also means that $\bar{\lambda}_i$ is σ_i-optimal. With the provisions above, the same holds for d_i (or it projection); that is one can stop when both $\|d_i\|$ and ϵ_i are "small". Our particular implementation of this is

$$t^*\|d_i\| + \epsilon_i \leq \eta \max(1, |f_i^{rec}|) \quad (2.6)$$

  where t^* is an appropriately chosen "large" scaling factor [25] and η is the required final relative accuracy (typically, $\eta =$ `1e-4`).
- As suggested in [56] (and in [3] in a different context), one could also use the deflection parameter α_i in a different way: not to change the gradient, but the point where it is evaluated. That is, for the recursive formulæ

$$\hat{\lambda}_i = \alpha_i\lambda_i + (1-\alpha_i)\hat{\lambda}_{i-1} \quad , \quad \hat{f}_i = \alpha_i f_i + (1-\alpha_i)\hat{f}_{i-1}$$

  with $(\hat{\lambda}_0, \hat{f}_0) = (0,0)$, it is obvious that $\hat{f}_i \geq f(\hat{\lambda}_i)$ for all i. One can then compute an approximation of the linearization error of d_i with respect to $\hat{\lambda}_i$ simply as

$$\hat{\epsilon}_i = \hat{\epsilon}_i(\hat{\lambda}_i) = \alpha_i\hat{\sigma}_i(\hat{\lambda}_i) + (1-\alpha_i)\hat{\epsilon}_{i-1}(\hat{\lambda}_i)$$

(with $\hat{\epsilon}_1(\hat{\lambda}_1) = \hat{\sigma}_1(\hat{\lambda}_1)$ and $\hat{\sigma}_i(\hat{\lambda}_i) = \hat{f}_i - [f_i + \langle g_i, \hat{\lambda}_i - \lambda_i \rangle]$), and keep it updated with the usual information transport property, this time reading

$$\hat{\epsilon}_i(\hat{\lambda}_{i+1}) = \hat{\epsilon}_i(\hat{\lambda}_i) - (\hat{f}_i - \hat{f}_{i+1}) - \langle d_i \, , \, \hat{\lambda}_{i+1} - \hat{\lambda}_i \rangle \ .$$

It is then immediate to prove that $d_i \in \partial_{\hat{\epsilon}_i} f(\hat{\lambda}_i)$ for all i, which allows to also employ the alternative stopping criterion

$$t^* \|d_i\| + \hat{\epsilon}_i \leq \eta \max(1, |f_i^{rec}|) \ . \tag{2.7}$$

The stopping conditions (2.6) and (2.7) are hardly if ever satisfied in practice. In our numerical experiments the algorithm always stops when either $f_i - f_*$ is "small" or (most often) the pre-set iterations limit is reached. However, we used (2.7) in PDSM, since all the terms involved in the formula have to be computed anyway (see the Appendix) and, consequently, testing (2.7) comes for free. For all the other approaches we only used (2.6), for again in most cases $\|d_i\|$ and ϵ_i are required anyway in the SR and/or the DR.

- For the incremental SG, we arrange outer and inner iterations in a simple pattern: we perform one outer iteration, followed by $|\mathcal{K}|$ inner iterations, one for each of the different components. Note that one of the components is the linear one corresponding to the RHS, which is treated as all the others. As suggested in [9, 60], we randomize the order in which the components are chosen, with the random permutation being changed at every outer iteration. We experimented with different ratios between inner and outer iterations but the results were inconclusive, with the simple approach being in general the best one. Furthermore, this means that a group of $|\mathcal{K}| + 1$ consecutive iterations (one outer, the other inner) costs, at least as far as the subproblem solution is concerned, as much as two full iterations. This is useful when comparing the running time of the approaches, as discussed in section 3.2.
- In the incremental SG, neither the inner steps nor the outer ones are deflected. Clearly, it makes little sense to combine together subgradients of different components. In principle, one could deflect the outer steps, but dealing with the direction of these steps in a different way would require keeping a copy of the direction of the outer steps and complicating the logic of the approach. For this reason, we have chosen to entirely avoid deflection in the incremental variants.
- Again for the incremental SG, (2.1) would require to compute the norm of $g_i \in \partial f(\lambda_i)$, but only the subgradient of one component is available. To make up for this, the theory [54] suggests to replace $\|g_i\|$ by the global Lipschitz constant L of f, yielding the modified stepsize rule

$$\nu_i = \beta_i \frac{\bar{f}_{p(i)} - f_i^{lev}}{\chi |\mathcal{K}| L^2} \tag{2.8}$$

where $p(i)$ the last outer step before i and χ is an arbitrary constant. In other words, one keeps the main part of the stepsize unchanged during sequences of inner iterations between two outer ones. In the same vein, in fact, in our experiments we used $\beta_i = \beta_{p(i)}$ and $f_i^{lev} = f_{p(i)}^{lev}$.
- A final important observation about the incremental case is that, when the active-set strategy is used (which, as we shall see, is crucial for performances), we update the active set only at full iterations. This choice is due to the fact

that this operation basically costs as much as one full iteration (see again section 3.2), and doing it, say, at every inner iteration would largely negate the advantage of having faster iterations. Yet, the fact that we perform one outer iteration every $|\mathcal{K}|$ inner ones means that the active set is updated "reasonably frequently". Updating the active set less frequently (say, $p \geq 1$ outer iterations) would be possible, but it has not shown to be computationally convenient in our application.

3. Numerical experiments. Nonsmooth optimization methods are widely used for bounding purposes within exact and heuristic approaches to large-scale IP, in particular with decomposable structure. For our tests we have selected as benchmark one of the most widespread structures in IP: the Fixed-Charge Multicommodity Capacitated Network Design (FC-MCND) problem [18] rapidly recalled below.

3.1. Lagrangian relaxations for FC-MCND. Given a directed network $G = (N, A)$, where N is the set of nodes and A is the set of arcs, we must satisfy the demands of a set of commodities K. Each commodity $k \in K$ is characterized by a *deficit vector* $b^k = [b^k_i]_{i \in N}$ indicating the net amount of flow required by each node: a negative deficit indicates a source, a positive deficit indicates a sink, and a zero deficit a transshipment node. Often, each commodity is an origin-destination pair (s_k, t_k) with an associated demand $d^k > 0$ that must flow between s_k and t_k, i.e., $b^k_i = -d^k$ if $i = s_k$, $b^k_i = d^k$ if $i = t_k$, and $b^k_i = 0$ otherwise. Each arc $(i, j) \in A$ can only be used if the corresponding *fixed cost* $f_{ij} > 0$ is paid, in which case the *mutual capacity* $u_{ij} > 0$ bounds the total amount of flow on (i, j), irrespective of the commodity. Also, *individual capacities* u^k_{ij} may be imposed for the maximum amount of flow of commodity k on arc (i, j); in the origin-destination case, for instance, one usually requires $u^k_{ij} \leq d^k$. Finally, the *routing cost* c^k_{ij} has to be paid for each unit of commodity k moving through (i, j). The problem consists in minimizing the sum of all costs while satisfying demand requirements and capacity constraints. The classical arc-flow formulation of the problem relies on variables x^k_{ij} for the amount of the flow of commodity k on arc $(i, j) \in A$ plus binary design variables y_{ij} for arc construction, resulting in

$$\min \textstyle\sum_{k \in K} \sum_{(i,j) \in A} c^k_{ij} x^k_{ij} + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.1}$$

$$\textstyle\sum_{(j,i) \in A} x^k_{ji} - \sum_{(i,j) \in A} x^k_{ij} = b^k_i \qquad i \in N\ ,\ k \in K \tag{3.2}$$

$$\textstyle\sum_{(i,j) \in A} x^k_{ij} \leq u_{ij} y_{ij} \qquad (i, j) \in A \tag{3.3}$$

$$x^k_{ij} \leq u^k_{ij} y_{ij} \qquad (i, j) \in A\ ,\ k \in K \tag{3.4}$$

$$x^k_{ij} \geq 0 \qquad (i, j) \in A\ ,\ k \in K \tag{3.5}$$

$$y_{ij} \in \{0, 1\} \qquad (i, j) \in A \tag{3.6}$$

For our tests we have employed the two main Lagrangian relaxations of the problem (3.1)–(3.6): the Flow Relaxation (FR) and the Knapsack Relaxation (KR).

In the former, one relaxes constraints (3.3)–(3.4) with multipliers $\lambda = [\alpha\,,\, \beta] = [\,\alpha_{ij}\,,\, \beta^k_{ij}\,]_{(i,j) \in A\,,\, k \in K} \geq 0$. This gives the objective function

$$\min \textstyle\sum_{(i,j) \in A} \sum_{k \in K} \big(c^k_{ij} + \alpha_{ij} + \beta^k_{ij}\big) x^k_{ij} + \sum_{(i,j) \in A} \big(f_{ij} - \alpha_{ij} u_{ij} - \sum_{k \in K} u^k_{ij} \beta^k_{ij}\big) y_{ij}$$

whose minimization subject to the remaining (3.2), (3.5) and (3.6) is easy, because of separability: it reduces to $|K|$ single-commodity Min-Cost Flow (MCF) problems, plus

$|A|$ trivial single-variable IPs. Thus, extremely efficient algorithms exist for solving the relaxation [32], especially when commodities are origin-destination pairs since then the MCFs actually are shortest path problems. Since constraints (3.2) only involve flow (continuous) variables, the Lagrangian dual provides the same bound as the continuous relaxation of (3.1)–(3.6). Note that the number of constraints (3.4) is typically rather large; these constraints being inequalities, this is the setting where active set techniques can be expected, and have indeed been proven, to be very competitive [30]. Clearly, checking the violation of these constraints, once a primal solution has been determined (see section 2.1.5), is immediate. It is also easy to realize that an estimate of the Lipschitz constant (useful for the incremental SM variants, see (2.8), as well as for PDSM, see (A.5)) is simply obtained as $L = \sqrt{\sum_{(ij)\in A}(u_{ij})^2 + \sum_{k\in K}\sum_{(ij)\in A}(u^k_{ij})^2}$. Note, however, that when the active set technique is used, the capacities entering the above formula are only those of the constraints whose indices are in the active set. This means that the Lipschitz constant actually changes (typically, increases) as the algorithm proceeds.

In the KR, one rather dualizes the flow conservation constraints (3.2) with multipliers $\lambda = [\lambda^k_i]_{i\in N, k\in K}$; this yields the objective function

$$\min \sum_{(i,j)\in A} \left(\sum_{k\in K}(c^k_{ij} - \lambda^k_i + \lambda^k_j)x^k_{ij} + f_{ij}y_{ij} \right) + \sum_{i\in K}\lambda^k_i b^k_i$$

whose minimization subject to the remaining (3.3)–(3.6) basically decomposes into $|A|$ very simple continuous knapsack problems, one for each arc, after which one can determine the optimal value of the unique integer variable y_{ij}. It can be shown that, due to the presence of the *strong forcing constraints* (3.4), the relaxation has the integrality property: hence, like in the previous case the Lagrangian dual has the same optimum value as the continuous relaxation. The number of multipliers, $|N| \cdot |K|$, is still rather large; however, these being equalities, it is rather unlikely that some of them is not going to be active at optimality, and therefore the active set technique is not likely to be effective. Furthermore, unlike in the FR, there are no (sign) constraints on the multipliers, and therefore no projection is needed. Finally, similarly to the FR it is easy to estimate the Lipschitz constant as $L = \sqrt{\sum_{k\in K}\sum_{i\in N}(L^k_i)^2}$, where $L^k_i = \max[\,|-b^k_i + \sum_{(ji)\in A} u^k_{ji}|\,,\,|-b^k_i - \sum_{(ij)\in A} u^k_{ij}|\,]$.

3.2. Experimental setup. We have implemented all the variants of SM within a general `C++` scheme for nonsmooth optimization developed by the authors along the years. The scheme is based on a pair of abstract (pure virtual) classes, `NDOSolver` and `FiOracle`, which establish the conceptual interface between the optimization algorithm (in our case, SM implemented in the class `Subgradient` derived by `NDOSolver`) and the oracle computing the function (in our case, the classes `FlowFiOrcl` and `KnapFiOrcl`, for FR and KR respectively, derived from `FiOracle`). Other implementations of nonsmooth approaches, such as Kelley's Cutting Plane method and different forms of (generalized) Bundle methods [3, 25, 30], were already available within the scheme. As already noted, the `Subgradient` class implementing the SM in turn relies on two external classes, `Stepsize` and `Deflection`, whereby the different SR (see section 2.1.1) and DR (see section 2.1.2) are implemented.

All solvers have been compiled with GNU `g++ 4.4.5` (with `-O3` optimization option) and ran single-threaded on a server with multiple Opteron 6174 processors (12 cores, 2.2 GHz), each with with 32 GB of RAM, under a i686 GNU/Linux operating system. To solve the FR, we have used MCF solvers from the `MCFClass` project, available at

`http://www.di.unipi.it/optimize/Software/MCF.html` ,
while solving the KR basically just required a sort and was coded directly. When comparing SM with other approaches we used `Cplex 12.5.0.1` to solve linear problems.

The numerical experiments have been performed on 80 randomly generated instances, arranged in 20 groups of 4 instances each. The first 8 groups are of small size. The remaining 12 groups are as fallows: the number of nodes and arcs are chosen as (20, 300), (30, 600), or (50, 1200), and for each of these, the number of commodities is chosen in the set {100, 200, 400, 800} (see Table 3.1). We refer to [30] for more details; the instances can be downloaded from
`http://www.di.unipi.it/optimize/Data/MMCF.html#Canad` .

A nontrivial issue about our experiments is how exactly one evaluates the performances of the different SM. Our choice has been to record the running time and the obtained lower bound of each variant with different iteration count limits. For all non-incremental SM, we (somewhat arbitrarily) choose that to be 100, 200, 500, 1000, 2000, 5000, and 10000 iterations. Clearly, for incremental SM a different choice is required because inner iterations are much faster; for these, we then choose iteration counts of 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000 and 1000000. The exact values of the iterations count are not particularly important, as we use them for charting the convergence speed of the different variants in terms of the all-important aspect of the time required to reach a certain gap with the (known) optimal value, and then report the whole chart. However, this approach has an issue in that instances of different sizes are naturally to be expected to require longer for each iteration, making it difficult to compute reasonably aggregated results. Fortunately, for our instances, this has proven to be particularly simple.

Indeed, we have observed that the charts for different instances (and the same SM variant, comprised the algorithmic parameters) were remarkably similar; even better, they became almost identical if the running time was divided by the product $|A| \cdot |K|$. Upon reflection, this is not particularly surprising. For the FR, one has to solve $|K|$ Lagrangian subproblems, each one being a shortest path on a graph with nonnegative arc costs, for which the standard approach has $O(|A| \log(|N|))$ complexity. For the KR, one rather has to solve $|A|$ continuous knapsack problems, which require $O(|K| \log(|K|))$. In other words, each full iteration of SM costs $O(|A| \cdot |K|)$ up to logarithmic factors; given the relatively limited range in which $|A|$ and $|K|$ vary in our experiments, any logarithmic factor is basically a constant. All the rest of the algorithm has a linear cost in the number of variables n, which is $(|A| + 1) \cdot |K|$ for the FR. With the active-set strategy that number is reduced, but again identification of new violated constraints has $O(|A| \cdot |K|)$ cost, as it requires forming the continuous solution $\bar{u}_i$ out of two relaxation solutions (again, an $O(|A| \cdot |K|)$ task) and scanning the full list of (3.4). Note that for incremental SM, the cost per iteration relative to the subproblem solution is just $O(|A|)$ (plus logarithmic factors); so, basically, the cost is dominated by that of forming the subgradient. This is why, as we shall see, active-set techniques are absolutely crucial in the incremental case. Also, this justifies the fact that updating the active set should only be done at full iterations. For the KR, $n = |N| \cdot |K|$, but $|N|$ is proportional to $|A|$ as the graphs are sparse; so, the cost per iteration (outside the computation of the Lagrangian relaxation) should be somewhat smaller than for the FR, but since no active-set technique is used, it is not surprising that the two tend to be similar. All in all, the iteration cost is dominated by the relaxation cost (as it usually is), which is roughly $O(|A| \cdot |K|)$; this explains

while the running time scales pretty much linearly in that quantity.

It is also remarkable that the convergence speed proved to be very similar for instances of sizes different by orders of magnitude: n varies from 9040 to 960000 for the FR and from 800 to 40000 for the KR, but for the same SM, the (scaled) convergence graphs were still pretty similar across the whole spectrum. This, too, is not entirely surprising. Indeed, the theoretical efficiency estimates of SM are typically independent on n, although they may depend on other parameters that can be indirectly influenced by it. Our experiments show that, at least for this application and SM we have tested, the practical behavior of the algorithms is in fact pretty much invariant with n. This confirms that SM can be especially promising for very large-scale problems, provided of course that the convergence speed is not abysmal.

Anyway, all this allowed us to compare the different SM variants, and the different settings of their (many) algorithmic parameters, by computing a graph showing the evolution of the (average) gap vs. the (average) *normalized time* across *all* the 80 instances of our test set. This has been instrumental in the tuning phase, discussed below. Also, this gave us a significant practical advantage for testing the incremental variants, that actually are randomized algorithms due to the selection of the random reshuffle of the components at each full iteration. Because of this, one should in principle run each instance several time and report averaged results among each run. However, because we report in one blow the aggregated results among all of the 8o instances, this is not necessary because any luck of draw is smoothed out by the large size of the instance set the results are averaged upon.

A final relevant aspect of our computational tests concerns the fact that the step-size rules of the form (2.1) are based on the knowledge of some (lower) approximation $\underline{f}$ to f_*. In order to avoid target-level approaches (unless when they are inherent in some specific SR), we have elected to work with a fixed $\underline{f}$. However, in order to cater for the different cases that would occur when using these techniques in IP, we have used two different configurations: in one we set $\underline{f} = f_*$, and in the other one $\underline{f}$ is 10% off of f_*, i.e., $\underline{f} = f_* - 0.1|f_*|$ (we denote the latter by "10%f_*"). Note that f_* is the optimal value of (1.1), which is a minimization problem: however, the Lagrangian dual (3.1)–(3.6) is rather a maximization problem. Indeed, in the implementation we have to change the sign of the objective function (and of the subgradients). Thus, using 10%f_* corresponds, in primal terms, to the fact that the available primal solution is 10% more costly than the best possible lower bound; in other words, even if the best possible bound is computed, the corresponding node in the enumeration tree cannot be fathomed. A gap of 10% between the upper and the lower bound is on the large side, but (unfortunately) not at all unreasonable in many applications. Conversely, using f_* corresponds to the case where a node in the enumeration tree can be fathomed by the bound (if it is computed accurately enough).

3.3. The tuning phase. We investigated a large number of variants, due to the almost combinatorial choices of the different main elements (SR, DR, `StepRes`, projection, active-set, ...). Each of them typically has some algorithmic parameters: consequently, the number of combinations of parameters to be examined has been very large. A specific recount of all the tested parameters, the ranges of values tested for each and the best combinations obtained is provided in the Appendix.

As discussed in the previous paragraph, each SM configuration gave rise to an aggregated convergence graph, as the ones shown in sections 3.4 and 3.5. To select the best configurations, the graphs were visually inspected. A possible issue here concerns the choice of the "best" convergence graph, in that conceptually one may

have configurations that perform significantly better at some stage of the algorithm (e.g., initial) but are not competitive at some other stage (e.g., final). Fortunately, this turned out not to be a serious concern. Basically, the chosen configurations were those that gave the best function value at the end of the algorithm. Occasionally, other configurations gave better results than the chosen one in the earlier stages of the algorithm on some subsets of the instances; usually the advantage was marginal at best, and only on a fraction of the cases, while the disadvantage in terms of final result was pronounced.

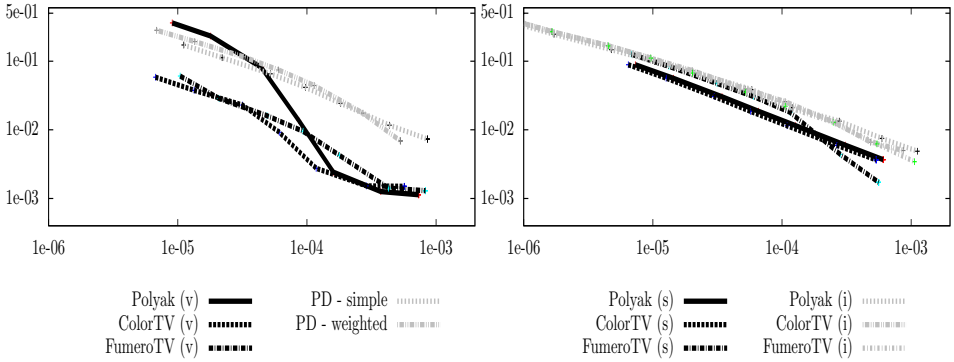
Although the total number of possible combinations was rather large, it turned out that only a relatively small set of parameters had a significant impact on the performances. Furthermore, in most of the cases, their effect was almost orthogonal to each other. This allowed us to effectively single out "robust" configurations for our test sets; for several of the parameters, the "optimal" choice has been unique across all instances, which may provide useful indications even for different problems. We refer to the Appendix for an in-depth discussion of the parameters, the tuning phase and the obtained results. All in all, our experience has been that, while it is true that SM requires a substantial amount of tuning, identifying the better performing settings was less difficult than it might, on the outset, have been.

3.4. Results for the FR. In this section we report the numerical results of SM relative to FR, using the optimal parameters detailed in the Appendix. Each variant is represented in Figures 3.1 and 3.2 by means of a graph, where the horizontal axis represents *normalized* total time (see section 3.2) and the vertical one the average gap, both in logarithmic scale. In the figures, we report results for all the three variants of SR, combined with the two variants of DR (simple subgradient, denoted by "(s)", and volume, denoted by "(v)"). We also report the combination of all SR with the incremental approach, denoted by "(i)" (this has no deflection, as discussed in section 2.2), and separately the two variants of PDSM, respectively with simple and weighted averages.

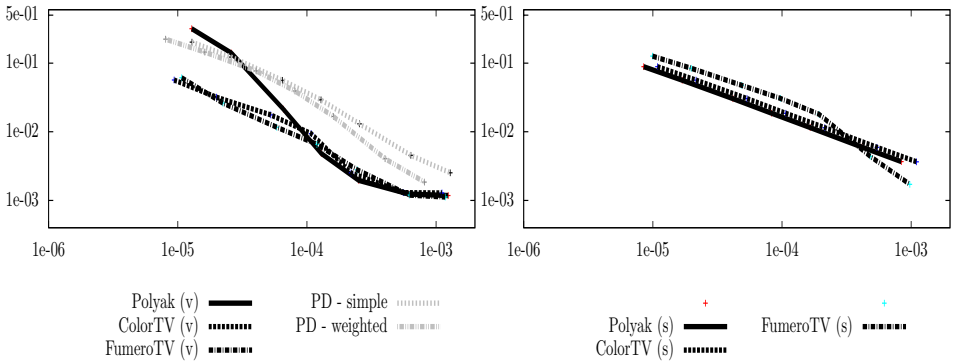
For clarity, we divide the graphs in four different quadrants, with the same scale on the horizontal and vertical axes to allow for comparison. In particular, in both figures, the upper two graphs (part (a)) depict results when the active-set strategy is used, and the lower two ones (part (b)) when it is not. Also, the leftmost graphs depict the approaches when deflection is used (`Volume`, and `Primal-Dual`) and the rightmost ones these where it is not (standard subgradient and incremental). Finally, Figure 3.1 reports the results when $\underline{f} = f_*$, while Figure 3.2 those where $\underline{f}$ is 10% off the optimal value ($10\% f_*$). We remark that PDSM do not use any knowledge about $\underline{f}$: this means that the corresponding convergence curves in the leftmost graphs of Figure 3.1 are exactly the same as these in the leftmost graphs of Figure 3.2. We also remark that we did not report the performances of incremental approaches without the active-set strategy (bottom right graphs of Figure 3.1 and Figure 3.2), because it was exceedingly slow. This is not surprising, because in the FR just scanning the whole subgradient has a cost comparable to that of solving all the subproblems; hence, just computing a subgradient is much more costly than solving just one component, thereby negating any advantage in having incremental iterations.

The following remarks can be made about the results.

- Deflected approaches are much more efficient than non-deflected ones; this is clearly visible in particular by comparing the same SR (left vs. right graphs). This requires properly choosing how to deflect, which may be problem-dependent, and (since the FR is constrained) which vectors among



(a) Results with the active-set strategy

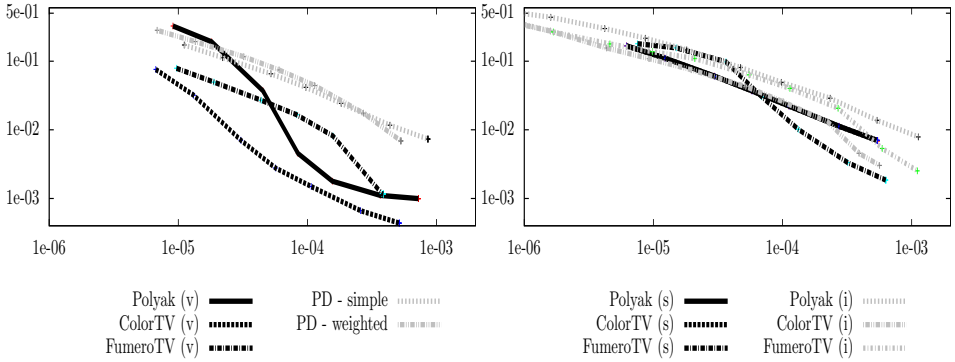


(b) Results without the active-set strategy

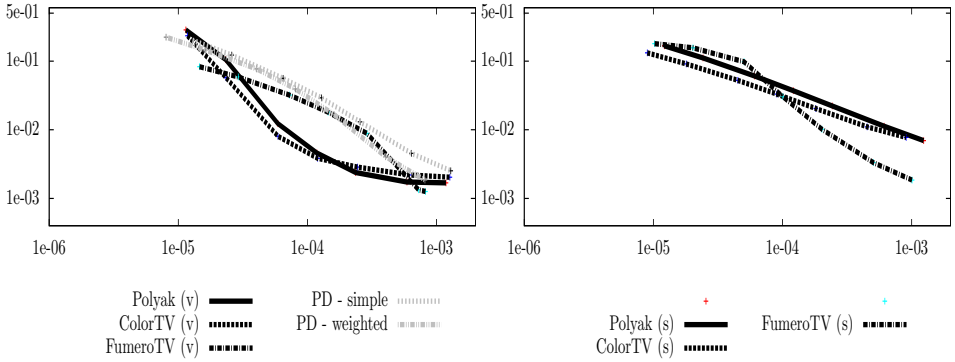
Figure 3.1: Results for the FR with lower bound f_*

d_i, d_{i-1} and g_i is better to project. However, as discussed in the Appendix, the different forms of projection have a limited impact on the performances, as long as any projection is performed, so deflection is most definitely the way to go.

- The incremental approaches are in general not competitive in our application. This is probably due to the combination of two factors. On the one hand, they are not deflected, which, as noticed right before, limit their efficiency. The second is that the number of variables is large, so that just handling one subgradient (computing or scanning it) requires much more time than solving one component. Thus, each iteration has a comparatively large "fixed cost" other than that of computing the function, which is independent on how many components are actually computed. While the active-set strategy manages to keep this cost somewhat lower, this is still not enough to make the incremental



(a) Results with the active-set strategy



(b) Results without the active-set strategy

Figure 3.2: Results for the FR with lower bound $10\% f_*$

approach convenient. This suggests that for the incremental approach to be interesting, the number of variables (relaxed constraints) should be "small", and/or the cost of solving each component should be comparatively "large" (although if the latter is very large then one may rather want to use algorithms with a much faster convergence rate, see section 3.6).

- PDSM are most often not competitive with the best "heuristic" ones. Their convergence behavior is very stable: on our bi-logarithmic graphs, the convergence curve of the simple average variant is remarkably linear, whereas that of the weighted average variant shows a clear (although not very pronounced) concave behavior indicating that convergence speed actually increases along the iterations. This is in line with the theoretical convergence estimates, and it can be considered a good result especially in view of the very limited effort required to tune these SM. Yet, finely tuned SM with other "heuristic" DR

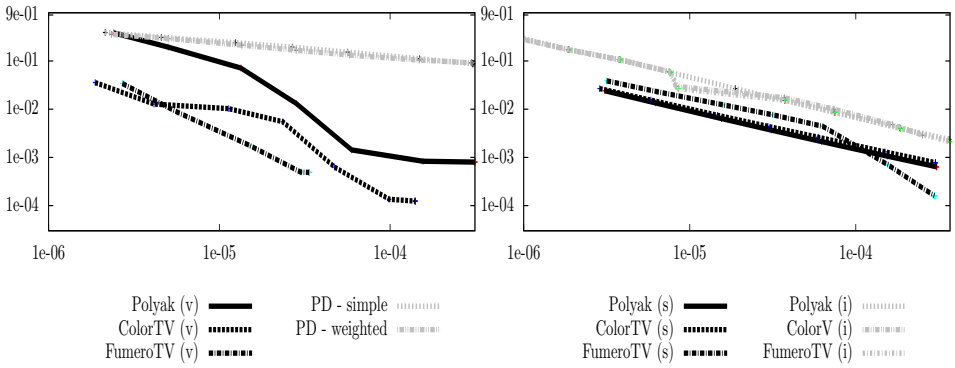
and SR can be significantly more effective. This may be due to the fact that PDSM do not make any use of available information, even if estimated, about the optimal value; this information is, instead, crucial for SR of the kind (2.1) that all other approaches use. It is therefore possible that it is exploiting this information that allows other SM to be more efficient in our application.

- Dynamic generation of the variables (active-set technique) is in general beneficial: SM are somewhat faster in performing the same number of iterations (the graphs Figure 3.1(a) and Figure 3.2(a) terminate somewhat more on the left than the corresponding ones in parts (b)), while the convergence rate is usually similar. There are exceptions, though, in both directions. For instance, Figure 3.2 shows that for ordinary SM (that is, except PDSM) the active-set approach can actually improve the convergence profile. The converse actually happens for PDSM, in both Figure 3.1 and Figure 3.2: the version without active-set converges faster than that using it. This is also somewhat unsurprising, for use of active-set techniques in the PSDM has never, to the best of our knowledge, been analyzed. It is therefore possible that a specific development would suggest changes in PDSM that could make it more efficient when the active-set technique is used.

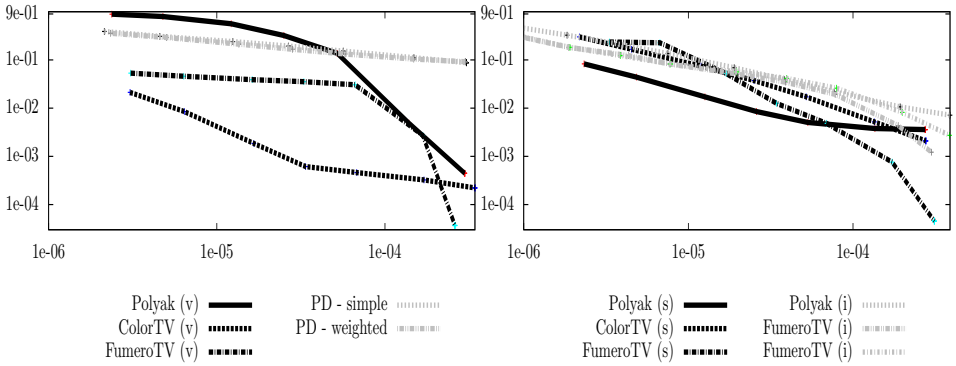
3.5. Results for the KR. We now report the results about the KR. These are summarized in Figure 3.3, with largely the same notation as for the FR case. However, in this case the active-set technique is not used, so we report just one figure: Figure 3.3(a) depicts the case with $\underline{f} = f_*$, like Figure 3.1, while Figure 3.3(b) that with $\underline{f} = 10\% f_*$, like Figure 3.2. Again, since PDSM do not use any bound to work, the corresponding curves in Figure 3.3(a) and Figure 3.3(b) are identical.

The following remarks can be made about the results:

- By and large, the same trends seen in the FR case show up here in terms of strong benefits of doing deflection and no benefits in doing incremental approaches.
- In this case, PDSM are even (much) less competitive. This may be due to the fact that PDSM have been developed under some sort of compactness assumption on the feasible set (see (A.1)), and actually use the (estimated) *diameter* of the set for optimally setting its algorithmic parameter. In the KR, the feasible set is in principle unbounded; in fact, the parameter corresponding to the diameter of the feasible set has to be experimentally tuned (see the Appendix). While this was true for the FR as well, in this case the *set of optimal solutions* is also unbounded, due to the fact that flow conservation constraints (3.2) are typically rank-deficient. This seems to significantly affect the practical behavior of PDSM.
- The left chart of Figure 3.3(a) shows a peculiar behavior of the `FumeroTV` rule: while it is the most efficient as it runs, it stops far before the maximal iteration limit because the steps become too small. As a consequence, it gets a far worse final gap than the other approaches (although it gets it quicker). This seems to be an issue with the rule, and no choice of the parameters we tested was able to avoid it. Interestingly, this only happens with deflection and $\underline{f} = f_*$; it does not with the standard subgradient method, nor with $\underline{f} = 10 f_*$, nor in any FR. It may be possible that some settings that we have not tested may avoid this behavior, but we elected not to do that; rather, this serves as a cautionary tale about the fact that heuristic rules, while possibly working well in many cases, may fail sometimes.



(a) Using lower bound f_*



(b) Using lower bound $10\% f_*$

Figure 3.3: Results for the KR

- Again in the same chart, the convergence graph of `ColorTV` is noticeably shorter than the others (save for `FumeroTV`). This is not a glitch: SM often attain the required gap of `1e-4` against the *known* lower bound f_*, at which point it is stopped. This might be considered "cheating": the algorithm does not stop with its means, but it is stopped due to externally provided information. However, this information is actually there in our application. Indeed, in several nodes of the enumeration tree, the value one wants to reach is *higher* than the minimum value of f (these are the nodes fathomed by the bound), which means that the SM (or any other approach, see section 3.6) is able to stop even sooner. This is particularly relevant for SM, whose standard stopping rules (see (2.6)) are scarcely effective; in all our tests, the algorithm very rarely stopped because of them. In fact, all other graphs (but for this) have roughly the same length: the algorithm almost always stops for having

exhausted the allotted maximum iterations.
- In general, approaching the KR provides better results than approaching the FR: better bounds are achieved more quickly. This confirms previous experiences [19].

3.6. Comparison with Cplex and Bundle methods. In order to properly position the obtained results in the wider contexts of computational methods for large-scale programs, in this section, we compare the best SM with two other approaches which provide the very same bound: solving the node-arc formulation (3.1)–(3.6) with a general-purpose LP solver (in particular, `Cplex`), and solving the Lagrangian duals of the FR and the KS using a Bundle method. The results are reported in Table 3.1. The following remarks are in order:

- For `Cplex`, an optimality gap of `1e-6` has been set, and this is considered "exact" as far as our application goes; thus, in the table no gap is reported. Tuning also has been performed, in that we have tested all of the (several) available methods, and ultimately we have selected the dual simplex algorithm as the one offering better performances; this was, however, almost always the algorithm chosen by the "automatic" setting. Also, the strong linking constraints (3.4) have been introduced in the formulation as *lazy constraints*—the equivalent of using the active set strategy in SM—which was absolutely crucial for performances (cf. [30, Table 4]). It is interesting to remark that, for the sake of fairness with the other methods, we experimented with passing to `Cplex` the optimal value of the problem in order to allow it to stop sooner. In order to do that we set an upper limit on the value of the objective function (using the parameter `CPX_PARAM_OBJULIM`) to $f_*(1 - \texttt{1e} - 4)$; since a dual simplex method is used, this should in principle allow `Cplex` to stop as soon as a dual solution with that accuracy is achieved. However, this turned out to be of no use. The reason is precisely the use of lazy constraints: `Cplex` separates them only when a feasible primal solution is attained, which is only at the end of the dual simplex. So, `Cplex` finds an accurate primal solution anyway, and cannot make use of the externally provided information. Not using the lazy constraints actually allowed `Cplex` to stop sooner when the information was provided, but it resulted in a hugely increased running time. By contrast, the other algorithms use infeasible primal solutions to do separation, and therefore do not suffer from this issue.
- The (generalized [25]) Bundle method that we used is implemented in the same `C++` framework as SM. A gap of `1e-4` was required, although (see, e.g., [30, Table 3 and Table 6]), unlike with SM, requiring substantially higher accuracy may only come at the cost of a comparatively minor increase in running times. Note that, for consistency, we passed the optimal value f_* to the Bundle algorithm, so that it could stop as soon as a solution with accuracy `1e-4` is attained; this always happened, so there is no need to report gaps in this case, either. We report results for two different variants of the Bundle method. For the FR, we used the fully disaggregated version with "easy component" and linear stabilization, denoted by DE-L in the table, that has been proven in [30]—after extensive tuning—to be the best option. It requires a large and complex master problem to be solved (with `Cplex`), which takes by far the largest fraction of running time; however, the number of iterations required to get convergence is highly reduced. For the KR, we performed extensive tuning, not discussed in details because it lies largely outside of the

scope of the paper, and we instead found the best Bundle variant to be the one that uses a fully aggregated master problem with quadratic stabilization (denoted by AK-Q in the table), where the master problem is solved with the specialized QP solver of [24].

- For SM, we report results corresponding to the best options identified in the previous phase. In particular, for the FR we have used `Volume` as DR and `Polyak` as SR (denoted by FVP in the table), with the active set strategy enabled, while for the KR we have still used `Volume` as DR, but `ColorTV` as the SR (denoted by KVC in the table). For both algorithms, we have set $\underline{f} = f_*$, and required a gap of `1e-4`. We also set an iteration limit of 5000, as it seemed to represent the best compromise between accuracy of the achieved solution and running time. For FVP, the algorithm invariably stopped because of the iteration limit, so we only report the final gap. KVC, instead, reasonably often—but not always—terminated before the iteration limit due to reaching the required accuracy. Thus, for this variant we report both the number of iterations and the final gap.

	dimension			`Cplex`	FVP		KVC			DE-L		AK-Q	
#	$\|N\|$	$\|A\|$	$\|K\|$	time	time	gap	time	iter	gap	time	iter	time	iter
1	20	226	40	0.05	1.76	1e-3	0.12	881	9e-5	0.09	12	0.25	1233
2	20	230	200	17.71	11.07	2e-3	5.39	4738	1e-4	16.34	30	10.44	8084
3	20	292	40	0.05	2.17	1e-3	0.10	602	1e-4	0.09	10	0.12	480
4	20	292	200	16.42	14.12	1e-3	6.08	4604	1e-4	12.54	28	8.50	5225
5	30	519	100	9.48	16.53	2e-3	3.15	3709	2e-4	10.05	34	8.05	7073
6	30	519	400	191.30	87.07	1e-3	20.62	4631	1e-4	80.28	25	57.42	6713
7	30	684	100	7.04	24.85	2e-3	3.27	3141	1e-4	10.90	53	5.03	3499
8	30	692	400	450.36	125.89	1e-3	26.16	4903	2e-4	188.33	32	82.67	9830
9	20	300	100	5.73	10.21	3e-3	2.52	5000	2e-4	7.36	35	3.62	5181
10	20	300	200	26.62	24.29	1e-3	6.65	5000	2e-4	19.96	30	10.10	6083
11	20	300	400	42.95	46.54	1e-3	17.45	4051	1e-4	16.77	26	38.18	5920
12	20	300	800	148.35	107.66	1e-3	25.42	3538	1e-4	38.32	23	33.76	3232
13	30	600	100	18.68	23.78	1e-3	6.13	4708	2e-4	7.93	42	11.16	6496
14	30	600	200	50.89	44.94	9e-4	14.09	3368	1e-4	8.93	34	25.59	3896
15	30	600	400	104.10	101.11	8e-4	20.98	3208	1e-4	11.51	22	30.55	3345
16	30	600	800	732.87	199.27	9e-4	52.98	3093	1e-4	61.28	25	84.30	3761
17	50	1200	100	51.91	56.21	1e-3	10.74	3580	1e-4	3.69	48	33.20	8985
18	50	1200	200	224.47	101.93	1e-3	30.42	4666	1e-4	34.27	43	59.89	7536
19	50	1200	400	833.57	227.48	9e-4	79.22	4499	1e-4	52.60	34	154.41	7630
20	50	1200	800	3749.56	468.26	8e-4	180.41	4900	1e-4	76.22	25	168.72	4174

Table 3.1: Comparison of the best SM with `Cplex` and Bundle methods

The table shows some interesting trends. While for small-scale instances, direct use of an LP solver is the best option, decomposition approaches become more and more competitive as the size grows. Often the Bundle method using "complex" master problems (DE-L) is the best option, as the trade-off between the size of the master problem and the number of iterations works in its favor. A further advantage of this approach is that, as shown in [30], with a comparatively minor increase in effort, one can get very high-quality dual solutions (with gaps of `1e-6`, and even `1e-12`), and the corresponding accurate optimal primal solutions. However, as the size of the

instances increase, the cost of the master problem becomes very high; thus, methods that use much cheaper master problems, even if they require many more iterations, can become competitive if a very low gap is not required. This is true sometimes for the aggregated Bundle method, but it is even more true for SM, in particular using the KR. Indeed, with only one exception (group 20), KVC is faster than AK-Q, while obtaining a roughly comparable gap. It is fair to remark, however, that KVC did not always attain the required `1e-4` accuracy, although it was always pretty close, whereas AK-Q always did. Yet, this confirms previous experience [13] that aggregated Bundle methods do not always attain significantly higher convergence rates than well-tuned SM, despite them collecting far more information and paying the corresponding price in terms of master problem time. Interestingly, in several cases (groups 2, 4–8, 10 and 12), SM obtain comparable gaps than DE-L in less time, often significantly so. However, this hinges on accurate selection of the many parameters of the SM; for instance, none of the FR-based SM approaches are competitive in this application. Yet, this confirms that appropriately designed and tuned SM could in principle be competitive with the state-of-the-art for efficiently computing (not too tight) bounds for hard, large-scale combinatorial problems.

4. Conclusion. We have computationally analyzed a large class of Subgradient Methods, covering many of the ones proposed in the literature so far, for the solution of large-scale Lagrangian Duals of hard combinatorial programs. The specific features of this application is that the number of variables is large, the computation of the function decomposes into many independent problems, and that only a relatively poor accuracy—typically of the order of `1e-4`—is required for the bound computation.

Our results show that, although the total number of variants (comprised the possible settings for the numerical algorithmic parameters) is rather large, it is not exceedingly difficult to find settings that work reasonably well across a large family of instances for any given application. Provided that the appropriate tuning is made, SM perform roughly as expected: while their global rate of convergence is far from being appealing, their very low cost per iteration—in particular, outside of the function computation—can make up for it as long as a relatively coarse bound is required.

Our interest in performing these experiments was partly about understanding the computational significance of the theory developed in [21]. In this sense, we can report that the ideas developed therein actually seem to have a significant computational impact. Deflecting is indeed crucial for good performances of a SM, and in the constrained case deflection and projection together do work better (see Table A.1). Interestingly, deflection-restricted approaches, developed for proving theoretical convergence of SM, actually seem to work well in practice in some cases (see Table A.2).

However, what mostly motivated our interest was the hope that two relatively recent additions to the arsenal of SM, namely incremental and primal-dual approaches, could significantly improve the performances with respect to more "traditional" ones. In this respect, and limited to the very specific instances and problems we have tested, against our expectations, we have to report that so far this has not happened. In hindsight, this might have been expected for incremental methods: the size of the variables space is large, while the subproblems are of very low complexity, which means that the "fixed cost" for each iteration (even if active-set techniques are applied where applicable) largely makes partial computation of the objective function not convenient. It is very likely that, within our target application, other instances exist where these trade-offs are significantly different, and therefore incremental methods are competi-

tive. Also, our results seem to suggest that incremental methods could significantly benefit from incorporating deflection techniques. This may require specific theoretical developments, e.g., along the lines of [69], but we have not further pursued this line because the incremental approach is anyway unlikely to be competitive for our specific instances.

As far as PDSM are concerned, our results show a very consistent behavior despite them having basically only one tunable parameter; hence, the approach appears to be very promising. Still, carefully tuned version of traditional SM can significantly outperform them in most scenarios. Our results seem to suggest that, therefore, there might be ways to improve PDSM, in practice and in theory, by taking into account the following aspects:

- exploiting information about the optimal value of the problem, if available;
- adapting the approach to cope with an active-set strategy;
- adapting the approach to cope with cases where the feasible set, and even worse the optimal set, is unbounded.

Advances in either of these aspects may make PDSM significantly more efficient in practice in our application, allowing practitioners to finally rely on a single robust, yet efficient, SM that only requires tuning of very few parameters. Unfortunately, it does not seem that this is already the case, at least for the instances at hand. However, our analysis will hopefully stimulate further research, that may ultimately bring about the necessary improvements.

A different line of research concerns the actual use of SM within enumerative approaches for the original large-scale, hard combinatorial programs. In such a framework, trading faster bound computation for lower bound quality can indeed improve the overall efficiency of the approach, but only if the right choices are made, which is far from being trivial. Furthermore, optimization of the Lagrangian function is required not just once, but as many times as there are nodes in the enumeration tree (or more). Hence, *reoptimization* techniques become crucial, whereby the information generated at the parent node is exploited to improve the solution time at its descendants. Which SM are more efficient in this context, in terms of the global running time of the enumerative algorithm rather than of any single bound computation, is a question that to the best of our knowledge has been scarcely (if ever) researched so far, and we intend to pursue it in the future.

Acknowledgements. The first author gratefully acknowledges the contribution of the Italian Ministry for University and Research under the PRIN 2012 Project 2012JXB3YF "Mixed-Integer Nonlinear Optimization: Approaches and Applications." The work of the second author has been supported by the Post-Doctoral Fellowship D.R. No 2718/201 (Regional Operative Program Calabria ESF 2007/2013–IV Axis Human Capital, Operative Objective M2, Action d.5) and the Interuniversity Attraction Poles Programme P7/36 "COMEX: combinatorial optimization metaheuristics & exact methods" of the Belgian Science Policy Office. The work of the third author has been supported by NSERC (Canada) under grant 184122-09.

Appendix A.

We now describe in details all the parameters of the SM that we have tested in our numerical results, comprised the different rules (SR, DF, ...), together with the results of the tuning phase. We immediately remark that for some parameters it is quite hard even to set reasonable ranges of values, i.e., to determine what is in general a "too large" or "too small" value. Our approach in this case has been to select the initial range heuristically, and then test it. If the best value consistently ended up

being at one extreme of the interval, this was taken as a suggestion that the best value might actually be larger/smaller than the heuristically chosen relevant extreme, and the interval enlarged accordingly. This of course hinges on the assumption that the behavior of the algorithm is somewhat "convex" in these parameters; while this is not necessarily true, it seems to be a reasonable assumption at least for most of them. Furthermore, for the vast majority of parameters a "convex" behavior has been verified experimentally, in that we almost never found the case where different settings "far apart" provided better performances than these "in the middle."

A.1. General parameters of SM. The following parameters are common to all variants of SM we tested, basically irrespective of the specific rules for choosing the stepsize, the deflection, and so on (although some specific combinations do not make sense).

- As discussed in section 2.1, it is possible to project one or more among the vectors g_i, d_{i-1} and d_i on tangent cone T_i of Λ at $\bar{\lambda}_i$. We denote by `pr` $\subseteq \{ g_i , d_{i-1} , d_i \}$ the subset of vectors on which projection is done; in all our tests, `pr` is constant and does not depend on the iteration. As already remarked, `pr` $= \{ g_i , d_{i-1} , d_i \}$ makes no sense as T_i is convex. Furthermore, when no deflection is done $d_i = g_i$ and therefore only `pr` $= \{ g_i \}$ and `pr` $= \emptyset$ make sense.
- Regarding the order in which the stepsize and the deflection are chosen, we denote by `sg` $\in \{$ drs , dr0 , srs , sr0 $\}$ the four possible schemes:
  - drs: deflection-restricted scheme with the safe rule (2.4);
  - dr0: deflection-restricted scheme with no safe rule;
  - srs: stepsize-restricted scheme with the safe rule (2.3);
  - sr0: stepsize-restricted scheme with no safe rule.

  Of course, drs and dr0 make no sense when no deflection is performed.
- We denote by χ the parameter used to adjust the Lipschitz constant L in the incremental case (see (2.8)). We tested the nine different values $\chi \in \{$ `1e-8`, `1e-7`, `1e-6`, `1e-5`, `1e-4`, `1e-3`, `1e-2`, `1e-1`, `1` $\}$.
- For constrained problems (in our case, the FR), it is possible to resort to the active-set strategy where the constraints are separated incrementally, thus keeping the size of the vectors smaller. One crucial decision in this case is how often separation is performed: doing it less often avoids some computations, but at the risk of ignoring possibly relevant information for some time. The simplest approach is to perform separation after a fixed number of iterations; we denote by s_1 the parameter dictating this value. We have restricted ourselves to only $s_1 \in \{0, 1\}$, i.e., either not using the active-set strategy at all or performing separation at every iteration. Note that separation is only performed (if ever) for the strong forcing constraints (3.4) (the many Lagrangian multipliers β), whereas the mutual capacity constraints (3.3) (the much less Lagrangian multipliers α) are all kept in the active set at all iterations.

A.2. Parameters of the SR. We now examine in details the parameters of the three stepsize rules `Polyak` [64], `ColorTV` [5, 6] and `FumeroTV` [33]. Since all of them correspond to the basic form (2.1), basically we are looking at different ways for determining β_i and f_i^{lev}.

Polyak SR. This is the simple case where β_i and f_i^{lev} are kept fixed at all iterations (i.e., they do not depend on i). Here, we exploit the fact that in our application we have a dependable lower bound $\underline{f} \leq f_*$ (or, anyway, to the value we want to reach)

and simply test the two cases $f^{lev} \in \{f_*, 10\% f_*\}$. As for the othe parameter, we tested $\beta \in \{\, 0.01\,,\, 0.1\,,\, 1\,,\, 1.5\,,\, 1.99\,\}$.

ColorTV SR. This rule is based on the improvement $\Delta f = \bar{f}_{i-1} - f_i$ of f and the scalar product $\langle g_i, d_i \rangle$ to estimate "how successful is a step has been." Note, however, that in deflection-restricted schemes (i.e., drs and dr0) the latter is not available, and we resort to using $\langle g_i, d_{i-1} \rangle$. Each iteration is marked with "color" as follows:

1. `green`, if $\langle g_i, d_i \rangle > \rho$ and $\Delta f \geq \rho \max\{|f_i^{\rm rec}|, 1\}$;
2. `yellow`, if $\langle g_i, d_i \rangle < \rho$ and $\Delta f \geq 0$;
3. `red`, otherwise,

where $\rho > 0$ is a tolerance. Intuitively, `green` is a "good" step possibly indicating that a larger ν_i may have been preferable, whereas `red` is a "bad" one suggesting that ν_i is too large. Given three parameters c_g, c_y and c_r, and denoting by n_g, n_y and n_r the number of *consecutive* iterations with color `green`, `yellow` and `red`, respectively, β_i is updated as follows:

1. If $n_g \geq c_g$ then set $\beta_i = \min\{\, 2\,,\, 2\beta_{i-1}\,\}$;
2. If $n_y \geq c_y$ then set $\beta_i = \min\{\, 2\,,\, 1.1\beta_{i-1}\,\}$;
3. If $n_r \geq c_r$ then then set $\beta_i = \max\{\, \texttt{5e-4}\,,\, 0.67\beta_{i-1}\,\}$;
4. if none of the above cases occur, then set $\beta_i = \beta_{i-1}$.

Note that, therefore, a potentially important parameter is the arbitrarily fixed value β_0. Also, the method includes a simple target-following scheme whereby if $f_i \leq 1.05 f_i^{lev}$ then $f_i^{lev} = f_i - 0.05 f_i^{lev}$. Note that this never happens for $f^{lev} = 10\% f_*$. The method therefore has several parameters; we kept $\rho =$ `1e-6` fixed, and we tested all combinations of $\beta_0 \in \{\, 0.01\,,\, 0.1\,,\, 1\,,\, 1.5\,,\, 1.99\,\}$, $c_g \in \{\, 1\,,\, 10\,,\, 50\,\}$, $c_y \in \{\, 50\,,\, 100\,,\, 400\,\}$, and $c_r \in \{\, 10\,,\, 20\,,\, 50\,\}$.

FumeroTV SR. This SR has a complex management of f_i^{lev} and β_i, motivated by experimental considerations [33], that is subdivided into two distinct phases. The switch between the two is an iteration counter r, that is increased each time there is no improvement in the function value. This counter is used to define the exponential function $\sigma(r) = e^{-0.6933(r/r_1)^{3.26}}$, where r_1 is a parameter; note that $\sigma(r_1) \approx 1/2$, which is how the two apparently weird numerical parameters have been selected. The function σ, which is clearly decreasing in r, is used in two ways. The first is to determine the maximum number of non-improving steps, which is the smallest integer r_2 such that $\sigma_\infty \geq \sigma(r_2)$, where the threshold $\sigma_\infty > 0$ is another parameter: given r_1 and σ_∞, r_2 can be obtained with a simple closed formula. The second is to construct at each iteration the value of f_i^{lev} as a convex combination of the known global lower bound (which, not incidentally, this algorithm specifically tailored for IP is the only one to explicitly use) and the current record value, i.e.,

$$f_i^{lev} = \sigma(r)\underline{f} + (1 - \sigma(r)) f_i^{rec} \quad .$$

In the first phase, when r varies, the threshold varies as well: as $\sigma(r)$ decreases when r grows, f_i^{lev} is kept closer and closer to f_i^{rec} as the algorithm proceeds. In the second phase ($r \geq r_2$), where r is no longer updated, $\sigma(r) = \sigma_\infty$.

The procedure for updating r (hence $\sigma(r)$, hence f_i^{lev}) and β_i proposed in [33] uses four further algorithmic parameters: a tolerance $\delta > 0$, two integer numbers η_1 and $\eta_2 \geq 1$, and the initial value $\beta_0 \in (0, 2)$. The procedure is divided in two phases, according to the fact that the iteration counter r (initialized to 0) is smaller or larger than the threshold r_2. Similarly to `ColorTV`, the rule keeps a record value $\bar{f}_i$ (similar, but not necessarily identical, to f_i^{rec}) and declares a "good" step whenever $f_i \leq \bar{f}_i - \delta \max\{|\bar{f}_i|, 1\}$, in which case $\bar{f}$ is updated to f_i. In either phase, the number

of consecutive "non-good" steps is counted. In the first phase, after $\bar{\eta}_2$ such steps r is increased by one, and β_i is updated as $\beta_i = \beta_{i-1}/(2\beta_{i-1}+1)$. In the second phase r is no longer updated: after every "good" step β_i is doubled, whereas after $\bar{\eta}_1$ "non good" steps β_i is halved.

In the tuning phase we tested the following values for the parameters:

- $\sigma_\infty \in \{$ `1e-4`, `1e-3`, `1e-2` $\}$;
- $\delta =$ `1e-6`;
- $r_1 \in \{ 10, 50, 100, 150, 200, 250, 300, 350 \}$;
- $\beta_0 \in \{ 0.01, 0.1, 1, 1.5, 1.99 \}$;
- $\eta_1 \in \{ 10, 50, 100, 150, 200, 250, 300, 350 \}$;
- $\eta_2 \in \{ 10, 50, 100, 150, 200 \}$.

A.3. Parameters of the DR. We now describe in details the three deflection rules that we have tested, i.e., `STSubgrad` [64], `Primal-Dual` [56] and `Volume` [5, 6]. Actually, not much is required to describe the first one: simply, $\alpha_i = 1$ for all i, and hence $d_i = g_i$, and the stability center $\bar{\lambda}_i$ is always updated to λ_{i+1}. This simplifies a number of issues: the deflection-restricted schemes do not apply, and only two possible projection schemes do. However, as we shall see, deflection substantially improves convergence in practice.

Primal-Dual. The PDSM is based on a sophisticated convergence analysis aimed at obtaining optimal a-priori complexity estimates; details of the theory are outside the scope of this paper, the interested reader being referred to [56]. One basic assumption of the approach is that the feasible set Λ is endowed with a *prox-function* $d(\lambda)$, and that one solves the modified form of (1.1)

$$\min\{ f(\lambda) \,:\, d(\lambda) \leq D \,,\, \lambda \in \Lambda \} \tag{A.1}$$

restricted upon a compact subset of the feasible region, where $D \geq 0$ is a parameter. As D is never directly used in the algorithm, but only need to be estimated to tune its parameters, this can always be assumed to be true if f has a minimum λ_*: in particular, we take as prox-function the standard Euclidian distance from the initial iterate, i.e., $d(\lambda) = \|\lambda - \lambda_0\|^2/2$, in which case D has to be an estimate of $\|\lambda_* - \lambda_0\|^2/2$. While obtaining it is not straightforward, this at worst introduces one algorithmic parameter that has to be tuned; this is, in fact, how we dealt with the issue, as detailed below.

However, it has to be remarked that PDSM are not, on the outset, based on a simple recurrence of the form (1.4); rather, given two sequences of weights $\{\upsilon_i\}$ and $\{\omega_i\}$, the next iterate is obtained as

$$\lambda_{i+1} = \operatorname{argmin}\Big\{ \langle \textstyle\sum_{k=1}^{i} \upsilon_k g_k \,,\, \lambda \rangle + \omega_i d(\lambda) \,:\, \lambda \in \Lambda \Big\} \ . \tag{A.2}$$

Yet, in the unconstrained case $\Lambda = \mathbb{R}^n$ problem (A.2) readily reduces to the simple form (1.4), as the following Lemma shows.

LEMMA A.1. *Assume $\Lambda = \mathbb{R}^n$, select $d(\lambda) = \|\lambda - \lambda_0\|^2/2$, fix $\lambda_i = \lambda_0$ for all $i \geq 0$ in* (1.4)*. By defining $\Delta_i = \sum_{k=1}^{i} \upsilon_k$, the following DR and SR*

$$\alpha_i = \upsilon_i/\Delta_i \ \ (\in [0,1]) \qquad \textit{and} \qquad \nu_i = \Delta_i/\omega_i \tag{A.3}$$

are such that λ_{i+1} produced by (A.2) *is the same produced by* (1.4) *and* (2.2)*.*

Proof. Under the assumptions, (A.2) is a strictly convex unconstrained quadratic problem, whose optimal solution is immediately available by the closed formula

$$\lambda_{i+1} = \lambda_0 - (1/\omega_i) \textstyle\sum_{k=1}^{i} \upsilon_k g_k \ . \tag{A.4}$$

This clearly is (1.4) under the SR in (A.3) provided that one shows that the DR in (A.3) produces

$$d_i = (\textstyle\sum_{k=1}^{i} \upsilon_k g_k)/\Delta_i \ .$$

This is indeed easy to show by induction. For $i = 1$ one immediately obtains $d_1 = g_1$. For the inductive case, one just has to note that

$$1 - \frac{\upsilon_{i+1}}{\Delta_{i+1}} = \frac{\Delta_{i+1} - \upsilon_{i+1}}{\Delta_{i+1}} = \frac{\Delta_i}{\Delta_{i+1}}$$

to obtain

$$d_{i+1} = \alpha_{i+1} g_{i+1} + (1 - \alpha_{i+1}) d_i = \frac{\upsilon_{i+1}}{\Delta_{i+1}} g_{i+1} + \frac{\Delta_i}{\Delta_{i+1}} \frac{\sum_{k=1}^{i} \upsilon_k g_k}{\Delta_i} = \frac{1}{\Delta_{i+1}} \sum_{k=1}^{i+1} \upsilon_k g_k$$

as desired. □

Interestingly, the same happens if simple sign constraints $\lambda \geq 0$ are present, which is what we actually have whenever $\Lambda \neq \mathbb{R}^n$.

LEMMA A.2. *If $\Lambda = \mathbb{R}^n_+$, the same conclusion as in Lemma A.1 hold after λ_{i+1} has been projected.*

Proof. It is easy to see that the optimal solution of (A.2) with $\Lambda = \mathbb{R}^n_+$ is equal to that with $\Lambda = \mathbb{R}^n$, i.e. (A.4), projected over $\mathbb{R}^n_+$. □

Therefore, the appropriate choices allow us to fit PDSM in our general scheme by just having both the DR and the SR implemented as in (A.3); in particular, the stability center $\bar{\lambda}_i$ is *never* updated, so that it always remains the initial λ_0. As far as the choices of υ_i and ω_i are concerned, we closely follow the suggestions in [56]. For the former, we tested both the `Simple averages` approach, corresponding to $\upsilon_i = 1$, and the `Weighted averages` approach where instead $\upsilon_i = 1/\|g_i\|$. For the latter, we set $\omega_i = \gamma\hat{\omega}_i$, where $\gamma > 0$ is a constant and

$$\hat{\omega}_0 = \hat{\omega}_1 = 1 \quad , \quad \hat{\omega}_i = \hat{\omega}_{i-1} + 1/\hat{\omega}_{i-1} \quad \text{for } i \geq 2 \ ,$$

which implies $\hat{\omega}_{i+1} = \sum_{k=0}^{i} 1/\hat{\omega}_k$. The analysis in [56] suggests settings for γ that provide the best possible theoretical convergence, i.e.,

$$\gamma = L/\sqrt{2D} \qquad \text{and} \qquad \gamma = 1/\sqrt{2D} \ , \tag{A.5}$$

for the `Simple averages` and `Weighted averages` scheme, respectively (being L is the Lipschitz constant of f). As already remarked we do not know the constant D, which is related to the distance between λ_0 and any optimal solution. There is, however, a parameter in SM that is somehow related to a similar concept: it is "t^*" in the stopping formulæ (2.6)–(2.7). Roughly speaking, that parameter estimates how far at most one can move along a subgradient $g_i \in \partial f(\lambda_i)$ *when λ_i is an approximately optimal solution*. The parameter (although it scarcely impacts the actual behavior of SM, see section 3.5) has been tuned, in particular because it is also used in the same way by Bundle methods, which do have an effective stopping criterion (see section 3.6); hence, one might take $D = (t^*)^2 L$ as an estimate. Yet, t^* is supposed to measure distance of λ_* from a "good" λ_i, whereas D has to measure it from the initial λ_0, which typically is not at all "good": hence, we introduce a further scaling factor $F > 0$. This finally leads to taking $\gamma = (F\sqrt{L})/(t^*\sqrt{2})$ for `Simple averages`, and $\gamma = F/(t^*\sqrt{2L})$ for `Weighted averagesl` F is then experimentally tuned. In general one would expect $F > 1$, and the results confirm this; however, to be on the safe side we tested all the values $F \in \{$ `1e-4`, `1e-3`, `1e-2`,`1e-1`, `1`, `1e1`, `1e2`, `1e3`, `1e4` $\}$.

Volume. In this DR, α_i is obtained as the optimal solution of a univariate quadratic problem. As suggested in [5], and somewhat differently from the original [6], we use exactly the "poorman's form" of the master problem of the proximal Bundle method

$$\min \left\{ \nu_{i-1} \left\| \alpha g_i + (1-\alpha) d_{i-1} \right\|^2 /2 + \alpha \sigma_i(\bar{\lambda}_i) + (1-\alpha)\epsilon_{i-1}(\bar{\lambda}_i) \ : \ \alpha \in [0,1] \right\} \quad \text{(A.6)}$$

where the linearization errors $\sigma_i(\bar{\lambda}_i)$ and $\epsilon_{i-1}(\bar{\lambda}_i)$ have been discussed in details in section 2.2. Note that we use as *stability weight* the stepsize ν_{i-1} of the previous iteration: this is justified by the fact that that term corresponds to the stepsize that one would do along the dual optimal solution in a Bundle method (e.g. [3, 5, 25]). It may be worth remarking that the dual of (A.6)

$$\min \left\{ \ \max\{ \, g_i d - \sigma_i(\bar{\lambda}_i) \, , \, d_{i-1} d - \epsilon_{i-1}(\bar{\lambda}_i) \, \} + \|d\|^2/(2\nu_{i-1}) \, \right\} \ , \qquad \text{(A.7)}$$

where d has to be interpreted as the displacement from the current point, i.e., $d = \lambda - \bar{\lambda}_i$, is closely tied to (A.2) in PDSM. The difference is that in (A.7) one uses two (approximate) subgradients, the current one and that corresponding to the direction (hence taking into account all the history of computation), whereas in (A.2) one uses only one (approximate) subgradient obtained as weighted average of the ones generated at previous iterations. Problem (A.6) is inexpensive, because without the constraint $\alpha \in [0,1]$ it has the closed-form solution

$$\alpha_i^* \ = \ \frac{\epsilon_{i-1}(\bar{\lambda}_i) - \sigma_i(\bar{\lambda}_i) - \nu_{i-1} d_{i-1}(g_i - d_{i-1})}{\nu_{i-1} \|g_i - d_{i-1}\|^2} \ ,$$

and thus one can obtain its optimal solution by simply projecting α_i^* over $[0,1]$. However, as suggested in [5, 6] we rather chose α_i in the more safeguarded way

$$\alpha_i = \left| \begin{array}{ll} \alpha_{i-1}/10 & \text{if } \alpha_i^* \leq \texttt{1e-8} \\ \min\{\tau_i \, , \, 1.0\} & \text{if } \alpha_i^* \geq 1 \\ \alpha_i^* & \text{otherwise} \end{array} \right.$$

where τ_i is initialized to τ_0, and each τ_p iterations is decreased multiplying it by $\tau_f < 1$, while ensuring that it remains larger than $\tau_{\min}$.

In our scheme, the DR also controls how the stability center is updated. For `ColorTV`, the choice is dictated by a parameter $m > 0$, akin that used in Bundle methods (but also in the SR `FumeroTV` and `ColorTV`) to estimate "sufficient decrease." In particular, if $\bar{f}_i - f_{i+1} \geq m \max\{1, |f_i^{ref}|\}$ a Serious Step occurs and $\bar{\lambda}_{i+1} = \lambda_{i+1}$, otherwise a Null Step takes place and $\bar{\lambda}_{i+1} = \bar{\lambda}_i$.

For the tuning phase we have selected the following values for the above parameters:

- $\tau_0 \in \{ \, 0.01 \, , \, 0.1 \, , \, 1 \, , \, 10 \, \}$;
- $\tau_p \in \{ \, 10 \, , \, 50 \, , \, 100 \, , \, 200 \, , \, 500 \, \}$;
- $\tau_f \in \{ \, 0.1 \, , \, 0.4 \, , \, 0.8 \, , \, 0.9 \, , \, 0.99 \, \}$;
- $\tau_{\min} \in \{$ `1e-4`, `1e-5` $\}$;
- $m \in \{ \, 0.01 \, , \, 0.1 \, \}$.

A.4. Detailed results of the tuning phase. The tuning phase required a substantial computational work, and a nontrivial analysis of the results. In the end, however, we were able to identify, for each of the combinations between SR and DR, the values of the algorithmic parameters that performed reliably better (among these

we tested). As already remarked section §3.3, one issue was that, with some parameter settings, SM start with a slow convergence rate but the behavior significantly improves along the iterations, or, conversely, start with a good convergence rate but the behavior deteriorates later on. In general, however, it has always been possible to find "robust" settings that provided the best (or close so) gap at termination, but were not too far from the best gaps even in all the other stages. These were the ones that we selected.

For the sake of clarity and conciseness, in Tables A.1 and A.2, we directly report the chosen values of the parameters for FR and KR, respectively, briefly remarking about the effect of each parameter and their relationships. The behavior of SM has shown to be pretty similar in the two cases $\underline{f} = f_*$ and $\underline{f} = 10\% f_*$ (see section 3.3); hence, the tables report the values for $\underline{f} = f_*$, indicating within square brackets these for the case $10\% f_*$ if they happen to be different. The tables focus on the combinations between the three SR and the two DR, plus the incremental case; the parameters of `Primal-Dual` variant are presented separately since the SR is combined with the DR.

Results for the FR. We start describing the results for FR; these are summarized in Table A.1, except for those settings that are constantly optimal. In particular, `STSubgrad` and `Incremental` have better performances with $\text{pr} = \{\text{g}_\text{i}\}$, irrespective of the SR. This is not surprising as, since they do not use deflection, this simply says that projecting (in the only possible way) is better than not doing it, a hardly unexpected outcome. On the other hand, for `Volume` the optimal setting of the projection does depend on the SR, although it has to be noted that $\text{pr} = \{\text{d}_\text{i}\}$ and $\text{pr} = \{\text{d}_{\text{i}-1}\}$ are most often very close to each other. All the other parameters of `Volume` depend on the SR (although the stepsize-restricted scheme with no safe rule is often good), except $\tau_{\min}$ and m that are always best set to `1e-4` and 0.1, respectively. Another interesting observation is that, while `Volume` does have several parameters, it does seem that they operate quite independently of each other, as changing one of them always has a similar effect irrespective of the others. We also mention that for `ColorTV` the parameters c_y and c_r have little impact on the performance, whereas c_g plays an important role and it significantly influences the quality of the results. As for `FumeroTV`, σ_∞ and η_2 have hardly any impact, and we arbitrarily set them to `1e-4` and 50, respectively.

In PDSM, the only crucial value is F, used to compute the optimal value of γ in (A.5). We found its best value to be `1e2` and `1e3` for `Simple averages` and `Weighted averages`, respectively. The choice has a large impact on performances, which significantly worsen for values far from these.

Results for the KR. The best parameters for the KR are reported in Table A.2. Clearly, projection does not apply here, since the problem is unconstrained. Although the best values are in general different from the FR, confirming the (unfortunate) need for problem-specific parameter tuning, similar observations as in that case can be made. For instance, for `Volume`, the parameters were still more or less independent from each other, and $\tau_{\min}$ and m were still hardly impacting, with the values `1e-4` and 0.1 still very adequate. For `ColorTV`, results are again quite stable varying c_y. Yet, differences can be noted: for instance, for FR c_g is clearly the most significant parameter and dictates most of the performance variations, while for the KR the relationship between the two parameters c_r and c_g and the results is less clear. Similarly, for `FumeroTV` some settings are conserved: σ_∞ and η_2 have very little effect and can be set to `1e-4` and 50, respectively. In other cases, though, differences show off: for

	Polyak	ColorTV				FumeroTV		
	β_i	β_0	c_g	c_y	c_r	β_0	r_1	η_1
Volume	0.01	0.1	50	400	50 [10]	0.1	150 [100]	50
τ_0	10	1				1		
τ_f	.8	.8				.8 [.9]		
τ_i	200 [100]	100				100 [200]		
pr	$\{d_i, d_{i-1}\}$ $[d_{i-1}]$	g_i				g_i $[d_{i-1}]$		
sg	$sr0$ $[srs]$	$sr0$				$sr0$ $[drs]$		
STSubgrad	1.5	1.5 [0.01]	1	50	50	1.99 [1.5]	200 [50]	250 [150]
sg	$sr0$	$sr0$ $[srs]$				$sr0$		
Incremental	1.5 [0.1]	1.99 [1.5]	50 [10]	100 [400]	50	1.99 [1.5]	300 [50]	300 [100]
χ	`1e-3` [`1e-2`]	`1e-3`				`1e-3`		
sg	$sr0$ $[srs]$	$sr0$ $[srs]$				$sr0$		

Table A.1: Optimal parameters for the *Flow Relaxation*

instance the parameters η_1, r_1 and β_0 were more independent on each other than in the FR.

The parameters of `Primal-Dual` showed to be quite independent from the underlying Lagrangian approach, with the best value of F still being `1e2` for the simple averages case and `1e3` for the weighted average one. This confirms the higher overall robustness of the approach.

REFERENCES

[1] M. Ahookhosh. Optimal subgradient algorithms with application to large-scale linear inverse problems. Technical report, Optimization Online, 2014.

[2] K.M. Anstreicher and L.A. Wolsey. Two "well-known" properties of subgradient optimization. *Mathematical Programming*, 120(1):213–220, 2009.

[3] A. Astorino, A. Frangioni, A. Fuduli, and E. Gorgone. A nonmonotone proximal bundle method with (potentially) continuous step decisions. *SIAM Journal on Optimization*, 23(3):1784–1809, 2013.

[4] L. Bacaud, C. Lemaréchal, A. Renaud, and C. Sagastizábal. Bundle methods in stochastic optimal power management: A disaggregated approach using preconditioners. *Computational Optimization and Applications*, 20:227–244, 2001.

[5] L. Bahiense, N. Maculan, and C. Sagastizábal. The volume algorithm revisited: relation with bundle methods. *Mathematical Programming*, 94(1):41–70, 2002.

[6] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[7] A. Beck and M. Teboulle. Smoothing and first order methods: a unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

[8] H. Ben Amor, J. Desrosiers, and A. Frangioni. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics*, 157(6):1167–1184, 2009.

[9] D.P. Bertsekas and A. Nedić. Incremental subgradient methods for nondifferentiable optimization. *SIAM J. on Optimization*, 12(1):109–138, 2001.

		Polyak	ColorTV				FumeroTV		
		β_i	β_0	c_g	c_y	c_r	β_0	r_1	η_1
		0.1	0.1	50	50	50	0.1	10	10
						[10]		[50]	[50]
Volume	τ_0	1	1				1[10]		
	τ_f	.9 [.8]	.8 [.9]				.99 [.8]		
	τ_i	50	100 [50]				50 [200]		
	sg	$dr0$ [srs]	$dr0$ [srs]				$dr0$ [drs]		
STSubgrad		1.5	.01	50	50	50	1.99	50	250
		[.1]						[10]	[200]
	sg	$sr0$	$sr0$				$sr0$		
Incremental		1.5	1	50	100	50	1.5	100	100
		[.1]	[.01]	[10]			[1]	[10]	
	χ	`1e-5` [`1e-6`]	`1e-5`				`1e-5`		
	sg	$sr0$	srs				$sr0$		

Table A.2: Optimal parameters for the *Knapsack Relaxation*

[10] A. Borghetti, A. Frangioni, F. Lacalandra, and C.A. Nucci. Lagrangian heuristics based on disaggregated bundle methods for hydrothermal unit commitment. *IEEE Transactions on Power Systems*, 18(1):313–323, February 2003.

[11] R.I. Bot and C. Hendrich. A variable smoothing algorithm for solving convex optimization problems. *TOP*, 2014.

[12] U. Brännlund. A genralized subgradient method with relaxation step. *Mathematical Programming*, 71:207–219, 1995.

[13] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.

[14] P. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study*, 3:26–34, 1975.

[15] P. Cappanera and A. Frangioni. Symmetric and asymmetric parallelization of a cost-decomposition algorithm for multi-commodity flow problems. *INFORMS Journal on Computing*, 15(4):369–384, 2003.

[16] Y. Censor, R. Davidi, G.T. Herman, R.W. Schulte, and L. Tetruashvili. Projected subgradient minimization versus superiorization. *Journal on Optimization Theory and Applications*, 160(3):730–747, 2014.

[17] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[18] T.G. Crainic, A. Frangioni, and B. Gendron. Multicommodity capacitated network design. *Telecommunications Network Planning*, pages 1–19, 1999.

[19] T.G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design problems. *Discrete Applied Mathematics*, 112:73–99, 2001.

[20] A. Crema, M. Loreto, and M. Raydan. Spectral projected subgradient with a momentum term for the lagrangean dual approach. *Computers & Operations Research*, 34:31743186, 2007.

[21] G. d'Antonio and A. Frangioni. Convergence analysis of deflected conditional approximate subgradient methods. *SIAM Journal on Optimization*, 20(1):357–386, 2009.

[22] O. du Merle, J.-L. Goffin, and J.-P. Vial. On improvements to the analytic center cutting plane method. *Computational Optimization and Applications*, 11:37–52, 1998.

[23] S. Feltenmark and K. Kiwiel. Dual applications of proximal bundle methods, including lagrangian relaxation of nonconvex problems. *SIAM Journal on Optimization*, 10(3):697–

721, 2000.

[24] A. Frangioni. Solving semidefinite quadratic problems within nonsmooth optimization algorithms. *Computers & Operations Research*, 21:1099–1118, 1996.

[25] A. Frangioni. Generalized bundle methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002.

[26] A. Frangioni. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139:163–193, 2005.

[27] A. Frangioni and G. Gallo. A bundle type dual-ascent approach to linear multicommodity min cost flow problems. *INFORMS Journal on Computing*, 11(4):370–393, 1999.

[28] A. Frangioni and B. Gendron. 0-1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics*, 157(6):1229–1241, 2009.

[29] A. Frangioni and B. Gendron. A stabilized structured dantzig-wolfe decomposition method. *Mathematical Programming*, 140:45–76, 2013.

[30] A. Frangioni and E. Gorgone. Generalized bundle methods for sum-functions with "easy" components: Applications to multicommodity network design. *Mathematical Programming*, 145(1):133–161, 2014.

[31] A. Frangioni, A. Lodi, and G. Rinaldi. New approaches for optimizing over the semimetric polytope. *Mathematical Programming*, 104(2-3):375–388, 2005.

[32] A. Frangioni and A. Manca. A computational study of cost reoptimization for min cost flow problems. *INFORMS Journal on Computing*, 18(1):61–70, 2006.

[33] F. Fumero. A modified subgradient algorithm for lagrangean relaxation. *Computers and Operations Research*, 28(1):33–52, 2001.

[34] A.M. Geoffrion. Lagrangian relaxation and its uses in iteger programming. *Mathematical Programming Study*, 2:82–114, 1974.

[35] J. Gondzio, P. González-Brevis, and P. Munari. New developments in the primaldual column generation technique. *European Journal of Operational Research*, 224(1):41–51, 2013.

[36] S. Görtz and A. Klose. A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. *INFORMS Journal on Computing*, 24(4):597610, 2012.

[37] M. Guignard. Efficient cuts in lagrangean 'relax-and-cut' schemes. *European Journal of Operational Research*, 105:216–223, 1998.

[38] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[39] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii,. *Mathematical Programming*, 1:6–25, 1971.

[40] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms* II—*Advanced Theory and Bundle Methods*, volume 306 of *Grundlehren Math. Wiss.* Springer-Verlag, New York, 1993.

[41] M. Ito and M. Fukuda. A family of subgradient-based methods for convex optimization problems in a unifying framework. Technical report, Optimization Online, 2014.

[42] K.L. Jones, I.J. Lustig, J.M. Farwolden, and W.B. Powell. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming*, 62:95–117, 1993.

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

[44] K. Kiwiel, T. Larsson, and P.O. Lindberg. The efficiency of ballstep subgradient level methods for convex optimization. *Mathematics of Operation Research*, 23:237–254, 1999.

[45] K.C. Kiwiel. Convergence of approximate and incremental subgradient methods for convex optimization. *SIAM Journal on Optimization*, 14(3):807–840, 2003.

[46] K.C. Kiwiel and J.L. Goffin. Convergence of a simple subgradient level method. *Mathematical Programming*, 85(4):207–211, 1999.

[47] G. Lan and Y. Zhou. Approximation accuracy, gradient methods, and error bound for structured convex optimization. Technical report, University of Florida, 2014.

[48] T. Larsson, M. Patriksson, and A.-B. Strömberg. Conditional subgradient optimization – theory and applications. *European Journal of Operational Research*, 88(2):382–403, 1996.

[49] T. Larsson, M. Patriksson, and A.-B. Strömberg. Ergodic, primal convergence in dual subgradient schemes for convex programming. *Mathematical Programming*, 86:283–312, 1999.

[50] C. Lemaréchal. An extension of Davidon methods to nondifferentiable problems. In M.L. Balinski and P. Wolfe, editors, *Nondifferentiable optimization*, volume 3 of *Mathematical Programming Study*, pages 95–109. North-Holland, Amsterdam, 1975.

[51] C. Lemaréchal and A. Renaud. A geometric study of duality gaps, with applications. *Mathematical Programming*, 90:399–427, 2001.

[52] J.J. Moreau. Proximite et dualite dans un espace hilbertien. *Bull. Soc. Math. Fr.*, 93:273299, 1965.

[53] I. Necoara and J.A.K. Suykens. Application of a smoothing technique to decomposition in convex optimization. *IEEE Transactions on Automatic Control*, 53(11):2674–2679, 2008.
[54] A. Nedic and D. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *Mathematical Programming*, 120:221–259, 2009.
[55] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.
[56] Y. Nesterov. Primal-dual subgradient methods for convex optimization. *Siam J. Optim.*, 12:109–138, 2001.
[57] Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16:235–249, 2005.
[58] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.
[59] Y. Nesterov. Universal gradient methods for convex optimization problems. *Mathematical Programming*, 2014.
[60] E.S.H. Neto and A.R. De Pierro. Incremental subgradients for constrained convex optimization: A unified framework and new methods. *SIAM Journal on Optimization*, 20(3):1547–1572, 2009.
[61] A. Ouorou. A proximal cutting plane method using chebychev center for nonsmooth convex optimization. *Mathematical Programming*, 119(2):239–271, 2009.
[62] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.
[63] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *Siam Review*, 33(1):60–100, 1991.
[64] B.T. Polyak. Minimization of unsmooth functionals. *Zh.Vychisl.Mat.Fiz.*, 9(3):509–521, 1969.
[65] B.T. Sherali, B.T. Choi, and C.H. Tuncbilek. A variable target value method for nondifferentiable optimization. *Operations Research Letters*, 26:1–8, 2000.
[66] B.T. Sherali and C. Lim. On embedding the volume algorithm in a variable target value method. *Operations Research Letters*, 32:455462, 2004.
[67] N.Z. Shor. *Minimization methods for nondifferentiable functions*. Springer-Verlag, Berlin, 1985.
[68] M.V. Solodov and S.K. Zavriev. Error stability properties of generalized gradient-type algorithms. *Journal of Optimization Theory and Applications*, 98(3):663–680, 1998.
[69] P. Tseng. Conditional gradient sliding for convex optimization. *Mathematical Programming*, 125:263–295, 2010.
[70] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. In M.L. Balinski and P. Wolfe, editors, *Nondifferentiable optimization*, volume 3 of *Mathematical Programming Study*, pages 145–173. North-Holland, Amsterdam, 1975.