Sequential Minimal Optimization Extended to General Quadratic Programming

1st William Brendel
Snap Research
Snap Inc.
Los Angeles, USA
william.brendel@snap.com

2nd Luis Marujo
Snap Research
Snap Inc.
Los Angeles, USA
luis.marujo@snap.com

***Abstract*— Nearly two decades ago *Platt* introduced the sequential minimal optimization (SMO) algorithm [1] to solve the Support Vector Machine (SVM) dual quadratic programming optimization problem. SMO belongs to the family of Sequential Quadratic Programming (SQP) algorithms, and specifically aims to reduce the quadratic programming (QP) problem to its minimum at every iteration. As a result, SQP can be solved analytically and leads to an algorithm with linear time and space complexity. In 2005, *Fan et al.* [2] summarized most of the optimization strategies that can be applied to solve the SVM QP problem with SMO in the well known LIBSVM library. Presently, other QP problems with similar form as the SVM QP dual problem are solved using more time and space consuming algorithms than mobile computational requirements allow. This research strives to discern the conditions that allow SMO to be extended to other QP problems and its complexity of solving the minimal QP at each iteration.**

***Index Terms*—Quadratic optimization, Support vector machines, Natural language processing.**

I. Introduction

It is not surprising that problems involving quadratic programming (QP) occur frequently in the fields of Machine Learning, Natural Language Processing, Computer Graphics, and Computer Vision. Many applications are therefore formulated as graph theory problems, where typical objective functions to be optimized contain unary potentials related to nodes and binary potentials related to edges. For the last 50 years, graph theory has simply supported an ocean of computer science applications [3], [4]. To name a few examples, image segmentation as been formulated as minimum cut [5], [6], maximum weight independent set [7], maximum weight clique [8], [9] and minimum spanning tree [10] problems [11]. Multi-object tracking has been formulated as maximum weight independent set [12] and generalized minimum and maximum clique [13], [14] problems. In addition, SVM and Support Vector Clustering (SVC) formulation [15] can be viewed as an independent set problem [16], [17]. That said not all QP problem are directly related to graph theory, for example [18]–[22]. In the same way that graph problems can be reduced to one another, QP problems can be reformulated into problems that can be solved more efficiently. For example, maximum clique, maximum independent set and graph cut problems are linked to the vertex separator problem QP formulation [17]. Various cut problems are reformulated as spectral clustering problems that can be optimized via weighted kernel k-means algorithms [6], [23], achieving real-time computation performances. In *Tsang et al.*'s work [16], SVM and SVC are formulated as minimum enclosing ball problems, obtaining provably approximate optimal solutions in linear time, with a space complexity independent of the problem size. Since graph-theory problems (and their QP formulation) can be reduced to one another, many QP problems are re-formulated to optimize a simple quadratic objective function with linear constraints such as Eqn. (2). Consequently, it is natural to investigate the extent to which SMO can be generalized to solve such QP problems while considering high-speed and low-memory requirements typically found on mobile platforms.

A. Related Work to Sequential Minimal Optimization

SMO was originally designed to train a support vector machine that requires the solution of the very large SVM dual QP optimization problem [1] defined below:

$$\begin{aligned} \boldsymbol{\alpha}^\star \leftarrow \underset{\boldsymbol{\alpha}\in[0,\,\mathrm{C}]^n}{\arg\max} \quad & \mathcal{F}(\boldsymbol{\alpha}) : \mathbf{1}^\intercal\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^\intercal\mathbf{H}\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{y}^\intercal\boldsymbol{\alpha} = 0 \quad \text{with} \quad \mathbf{y}\in\{-1,\,1\}^n \\ & \mathbf{H}_{ij} = \mathbf{y}_i\mathbf{y}_j K(\mathbf{x}^i,\,\mathbf{x}^j) \;\; \text{and} \;\; \mathbf{x}^k \in I\!R^d \end{aligned} \tag{1}$$

SMO belongs to the family of SQP algorithms [24]. It breaks a large QP problem into a series of smaller QP problems, each of which optimizes a quadratic model of the objective subject to a linearization of the constraints. The method is equivalent to applying Newton's method to the KarushKuhnTucker (KKT) conditions of the QP problem. In SMO, the small QP problems involve only two variables. These small QP problems are solved analytically, thus avoiding the use of a time-consuming numerical QP optimization at each iteration. The amount of memory required for SMO is linear in the training set size n, which allows SMO to handle very large input sets.

SMO is not the only algorithm reducing the large SVM QP problem into a series of smaller problems [25], [26]. However, methods of [25], [26] are not scalable for very large training set as the small QP problems cannot fit into memory [1], which is still a problem nowadays, for example as the scientific community moved research tasks from MNIST to IMAGENET. A more radical approach than SMO is to avoid the QP altogether [27]. However, non-linear kernels still require the inversion of an $n \times n$ matrix. SMO is also closely related to a family of optimization algorithms called Bregman methods [28] or row-action methods [29]. These methods solve convex programming problems with linear constraints. They are iterative methods where each step projects the solution at each iteration onto each constraint. However, as stated in [1], unmodified Bregman methods cannot solve general non-convex QP problems with linear equality constraints directly. Other algorithms with analytical solutions for the small QP problems, like in [8], [30], could also be extended to solve the SVM QP formulation or more generally Eqn. (2). However, SMO presents a unique advantage: the direction in which the solution is updated at each iteration is extremely sparse (involving only two variables). Consequently, SMO reduces the time complexity of the overall process from $O(n^2)$ for standards algorithms to $O(n)$ while maintaining a linear space complexity. When the final solution is assumed to be sparse, SMO only focuses on optimizing the few dimensions that are not null. However, when the final solution is known to be dense, it optimizes only the best two dimensions at each iteration, leading to a sub-optimum solution. While we can always use other SQP algorithms [24], we offer in this paper a solution to escape poor local optimum. An extensive study on SMO applied to SVM can be found in [31]. We summarize the main contributions of the paper below.

B. Contributions

In the context of SVM, SMO was already extended, for example, to learn jointly support vectors and kernels [32]. This paper aims at extending SMO to the general form of QP described in Eqn. (2). More precisely: (1) We prove that the complexity of the smallest possible problem to be optimized at each iteration depends on the number of independent linear equality constraints. (2) We prove that optimizing the small problem at each iteration is equivalent to analytically solving a linear system and a bounded D-dimensional conic equation. (3) We extend the notion of a working set [2], [26] in order to escape local sub-optimal solutions. (4) We extend the notion of dimension selection [2], [30] to the general form of QP in Eqn. (2). (5) We show that SMO can be applied to QP problems other than the dual formulation of SVM and SVC, like document summarization. Below, we conclude our introduction with the set of notations and definitions used throughout the paper. For the remaining parts of the paper, we detail our contributions, focus on selected applications, demonstrate the results of our experiments, and finally present the conclusions.

C. Notations and definitions

Vectors and matrices are noted in bold, and matrices are in capital letters. $\mathbf{x}_i$ and $\mathbf{C}_{ij}$ are values at position i and j in vector $\mathbf{x}$ and matrix $\mathbf{C}$. $\mathbf{C}_j$ is the j^{th} column of $\mathbf{C}$, $\mathbf{C}_j^\intercal$ is the j^{th} row of $\mathbf{C}$. $\mathcal{F}$ is an objective function to be optimized with respect to $\mathbf{x}$, $\mathbf{x}^{(t)}$ represents the solution at iteration t and $\mathbf{x}^\star$ the final solution when the KKT conditions are met. In the remaining part of the paper we simplify the notation: $\mathcal{F} = \mathcal{F}(\mathbf{x})$, $\mathcal{F}^{(t)} = \mathcal{F}(\mathbf{x}^{(t)})$, $\mathcal{F}^\star = \mathcal{F}(\mathbf{x}^\star)$, $\delta\mathcal{F}^{(t)} = \delta\mathcal{F}^{(t)}(\boldsymbol{\delta}\mathbf{x})$ and $\boldsymbol{\delta}\mathbf{x} = \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}$. We denote by $\nabla\mathcal{F}$ the gradient of $\mathcal{F}$ with respect to $\mathbf{x}$ and by $\delta\mathcal{F}^{(t)} = \mathcal{F}^{(t+1)} - \mathcal{F}^{(t)} = \mathcal{F}(\mathbf{x}^{(t+1)}) - \mathcal{F}(\mathbf{x}^{(t)})$ the objective function of the small QP problem at iteration t. In the following $\mathbf{x}$, $\mathbf{x}^{(t)}$, $\mathbf{x}^\star$ and $\boldsymbol{\delta}\mathbf{x} \in I\!R^n$. $\mathbf{D}_\mathbf{x} = diag(\mathbf{x})$ is the diagonal matrix where $(\mathbf{D}_\mathbf{x})_{ii} = \mathbf{x}_i$. We define $\mathcal{S} = \{i_k\}_{k=1}^K$ as a set of K indices $i_k \in \{1, n\}$. We define the operator $(.)_\mathcal{S}$ so that $\mathbf{C}_\mathcal{S} = [\mathbf{C}_{i_1}, .., \mathbf{C}_{i_k}] \in I\!R^{m\times K}$ is the matrix made of K columns of $\mathbf{C}$, with $(\mathbf{C}_\mathcal{S})_k = \mathbf{C}_{i_k}$. Respectively $\mathbf{C}_{\mathcal{S}\mathcal{S}} = (\mathbf{C}_\mathcal{S}{}^\intercal)_\mathcal{S}{}^\intercal \in I\!R^{K\times K}$ and $\mathbf{x}_\mathcal{S} = [\mathbf{x}_{i_1}, .., \mathbf{x}_{i_K}]^\intercal \in I\!R^K$.

Definition 1. *We call $\mathcal{R}ow(\boldsymbol{C})$ the vector space spanned by the rows of $\boldsymbol{C}$, and rank$(\boldsymbol{C}) = |\mathcal{R}ow(\boldsymbol{C})|$ is the dimension $\mathcal{R}ow(\boldsymbol{C})$, i.e. the number of independent rows of $\boldsymbol{C}$.*

Definition 2. *We call $\mathcal{K}er(\boldsymbol{C}) = \{\boldsymbol{x} \in I\!R^n \mid \boldsymbol{Cx} = \boldsymbol{0}\}$ the null space of $\boldsymbol{C}$, and nullity$(\boldsymbol{C}) = |\mathcal{K}er(\boldsymbol{C})|$. Then $n = $ rank$(\boldsymbol{C})$ + nullity$(\boldsymbol{C})$.*

II. Problem Setup

We aim to generalize SMO to optimize the QP problem:

$$\mathbf{x}^\star \leftarrow \underset{\mathbf{x}\in[r,\,R]^n}{\text{optimize}} \quad \mathcal{F} : \mathbf{x}^\intercal\mathbf{A}\mathbf{x} + \mathbf{b}^\intercal\mathbf{x} \quad \text{s.t.} \quad \mathbf{C}\mathbf{x} = \mathbf{d} \tag{2}$$

where $\mathbf{C} \in I\!R^{m\times n}$ and $\mathbf{Cx} = \mathbf{d}$ forms a *"non-over-constrained"* system. Inequality constraints of the form $\mathbf{Cx} \leq \mathbf{d}$ can be directly added to the objective function $\mathcal{F}$ via Lagrangian multipliers [33], this solving a dual QP problem of the same form as Eqn. (2). The problem can also be reformulated using slack variables, leaving only equality constraints [33]. The iterative formulation of Eqn. (2) aims at solving:

$$\begin{aligned} \boldsymbol{\delta}\mathbf{x}^\star \leftarrow \underset{\boldsymbol{\delta}\mathbf{x}}{\text{optimize}} \quad & \delta\mathcal{F}^{(t)} : \boldsymbol{\delta}\mathbf{x}^\intercal\mathbf{A}\boldsymbol{\delta}\mathbf{x} + \nabla\mathcal{F}^{(t)\intercal}\boldsymbol{\delta}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{C}\boldsymbol{\delta}\mathbf{x} = \mathbf{0},\ \mathbf{x}^{(t)} + \boldsymbol{\delta}\mathbf{x} \in [r,\,R]^n \\ \text{and} \quad & \nabla\mathcal{F}^{(t)} = (\mathbf{A} + \mathbf{A}^\intercal)\mathbf{x}^{(t)} + \mathbf{b} \end{aligned} \tag{3}$$

where we assume that $\mathbf{x}^{(0)}$ is initialized so that $\mathbf{Cx}^{(0)} = \mathbf{d}$, and $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \boldsymbol{\delta}\mathbf{x}^\star$. The goal of SMO is to constrain $\boldsymbol{\delta}\mathbf{x}$ to be extremely sparse so that Eqn. (3) can be solved analytically. While traditional formulation would control the sparsity of $\boldsymbol{\delta}\mathbf{x}$ by incorporating an additional constraint like $\|\boldsymbol{\delta}\mathbf{x}\|_1 < \nu$, SMO explicitly controls the sparsity of $\boldsymbol{\delta}\mathbf{x}$ via a set $\mathcal{S}$ of K indices:

$$\begin{aligned} \boldsymbol{\delta}\mathbf{x}^\star_{\mathcal{S}^\star} \leftarrow \underset{\boldsymbol{\delta}\mathbf{x},\,\mathcal{S}}{\text{optimize}} \quad & \delta\mathcal{F}^{(t)}_\mathcal{S} : \boldsymbol{\delta}\mathbf{x}_\mathcal{S}{}^\intercal\mathbf{A}_{\mathcal{S}\mathcal{S}}\boldsymbol{\delta}\mathbf{x}_\mathcal{S} + \nabla\mathcal{F}^{(t)\intercal}_\mathcal{S}\boldsymbol{\delta}\mathbf{x}_\mathcal{S} \\ \text{s.t.} \quad & \mathbf{C}_\mathcal{S}\boldsymbol{\delta}\mathbf{x}_\mathcal{S} = \mathbf{0},\ \mathbf{x}^{(t)}_\mathcal{S} + \boldsymbol{\delta}\mathbf{x}_\mathcal{S} \in [r,\,R]^K \\ \text{and} \quad & \nabla\mathcal{F}^{(t)}_\mathcal{S} = ((\mathbf{A} + \mathbf{A}^\intercal)\mathbf{x}^{(t)} + \mathbf{b})_\mathcal{S} \end{aligned} \tag{4}$$

In [1], [2], the later system is first optimized with respect to $\mathcal{S}$, and then with respect to $\boldsymbol{\delta}\mathbf{x}$. The update rule is then $\forall\, i_k \in \mathcal{S}^\star$, $\mathbf{x}^{(t+1)}_{i_k} \leftarrow \mathbf{x}^{(t)}_{i_k} + (\boldsymbol{\delta}\mathbf{x}^\star_{\mathcal{S}^\star})_k$. In summary, designing a SMO algorithm involves 3 steps: we need (i) to fix the number of indices $K = |\mathcal{S}|$, (ii) to design a heuristic to select the best set of indices $\mathcal{S}^\star$, and (iii) to solve Eqn. (4) analytically with respect to $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}^\star}$ once $\mathcal{S}^\star$ is found.

III. Finding the Minimal QP Problem Size

The minimal QP size is directly related to K, the number of indices in $\boldsymbol{\mathcal{S}}$. In the context of solving the SVM dual Eqn. (1), [1] fixed $K = 2$. If we add a sparsity constraint $\mathbf{1}^{\mathsf{T}}\boldsymbol{\alpha} = \nu$ to Eqn. (1) to have fewer support vectors, the linear constraints are summarized by the system $\mathbf{C}\boldsymbol{\alpha} = \mathbf{d}$ with $\mathbf{C} = [\mathbf{y},\, \mathbf{1}]^{\mathsf{T}} = \left[\begin{smallmatrix} -1 & -1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \end{smallmatrix}\right]$ and $\mathbf{d} = [0,\, \nu]^{\mathsf{T}}$. However, if we maintain $K = 2$ and if we inadvertently pick the index set $\mathcal{S}$ such that $\mathbf{C}_{\mathcal{S}} = \left[\begin{smallmatrix} -1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$ then $\mathbf{C}_{\mathcal{S}}\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = \mathbf{0} \Rightarrow \boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = \mathbf{0}$, since the columns of $\mathbf{C}$ are orthogonal, hence independent. This means the iterative system becomes stationary and the solution may never converge to an optimum. Previous research [2] solved the issue by selecting $\mathcal{S} = \{i_1,\, i_2\}$ so that $\mathbf{y}_{i_1} = \mathbf{y}_{i_2}$, and extended the selection of K to any matrix $\mathbf{C}$.

Lemma 1. *Let K be the number indices in $\mathcal{S}$ needed to optimize Eqn.* (4)*. $rank(\boldsymbol{C}) \leq n - 1 \Rightarrow K \geq rank(\boldsymbol{C}) + 1$.*

Proof. $K \leq n$, hence $\mathrm{rank}(\mathbf{C}_{\mathcal{S}}) \leq \mathrm{rank}(\mathbf{C})$ as we may have removed some of the independent columns of $\mathbf{C}$ during the projection onto the dimensions selected by $\mathcal{S}$. This leads to $\mathrm{rank}(\mathbf{C}_{\mathcal{S}}) \leq \mathrm{rank}(\mathbf{C}) \leq K - 1$. Based on Def. 2, we have $\mathbf{C}_{\mathcal{S}} \in I\!R^{m \times K} \Rightarrow K = \mathrm{nullity}(\mathbf{C}_{\mathcal{S}}) + \mathrm{rank}(\mathbf{C}_{\mathcal{S}})$. Hence $\mathrm{nullity}(\mathbf{C}_{\mathcal{S}}) = K - \mathrm{rank}(\mathbf{C}_{\mathcal{S}}) \geq \mathrm{rank}(\mathbf{C}_{\mathcal{S}}) + 1 - \mathrm{rank}(\mathbf{C}_{\mathcal{S}})$ leading to $\mathrm{nullity}(\mathbf{C}_{\mathcal{S}}) \geq 1$. This means that $\forall \mathcal{S}$ with $|\mathcal{S}| = K$, $\exists\, \boldsymbol{\delta}\mathbf{x}^{\star}_{\mathcal{S}} \in \mathcal{K}er(\mathbf{C}_{\mathcal{S}})$ such that $\boldsymbol{\delta}\mathbf{x}^{\star}_{\mathcal{S}} \neq \mathbf{0}$ and $\boldsymbol{\delta}\mathbf{x}^{\star}_{\mathcal{S}}$ is an optimizer of Eqn. (4) for a fixed $\mathcal{S}$, i.e. the system Eqn. (4) is not stationary until it reaches an optimum. □

Note that when $\mathrm{rank}(\mathbf{C}) = n$, we can directly solve $\mathbf{x}^{\star} = \mathbf{C}^{-1}\mathbf{d}$ if $\forall i \in \{1,\, n\}$, $\mathbf{x}^{\star}_i \in [r,\, R]$ (otherwise no valid solution exists). When $\mathbf{C}$ is full rank, we can get $\mathbf{C}^{-1}$ analytically by using the Jordan-Gauss elimination method. In addition, we show in the supplemental material that Eqn. (4) can be reformulated so that $K \geq 1$ by parameterizing $\boldsymbol{\delta}\mathbf{x}$ as a linear combination of vectors spanning $\mathcal{K}er(\mathbf{C})$. However, it is computationally expensive when $|\mathcal{K}er(\mathbf{C})|$ is large.

IV. Dimension Selection and Working Set

Once we have chosen K, we need to find the set $\mathcal{S} = \{i_k\}_{k=1}^{K}$ before optimizing Eqn. (4) with respect to $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}}$. [2], [26] introduced the notion of "*working set*" related to the most violating pairs and second order information. Both are linked to the first and second order approximation of the objective function $\mathcal{F}$. [30] also chooses the dimensions to be optimized based on the highest gradient magnitude along a specific dimension. We propose the following heuristic to extend the notion of working set. At each iteration t, we maintain two sets, an active set $\mathcal{S}_A^{(t)}$ and an inactive set $\mathcal{S}_I^{(t)}$. Let $\nabla\mathcal{F}^{(t)}$ be the gradient of the objective function at time t, and $\eta^{\star}_i$ be the best potential displacement along the i^{th} direction of the gradient:

$$\eta^{\star}_i \leftarrow \underset{\eta_i}{\text{optimize}}\ \eta_i \nabla\mathcal{F}^{(t)}_i \Leftrightarrow w^{\star} \leftarrow \underset{w = r \text{ or } R}{\text{optimize}}\ (w - \mathbf{x}^{(t)}_i)\nabla\mathcal{F}^{(t)}_i$$
$$\text{with}\quad \eta^{\star}_i = w^{\star} - \mathbf{x}^{(t)}_i \quad\text{and}\quad \nabla\mathcal{F}^{(t)}_i = ((\mathbf{A} + \mathbf{A}^{\mathsf{T}})\mathbf{x}^{(t)} + \mathbf{b})_i \qquad (5)$$

Algorithm 1 Initializing $\mathcal{S}$ with the first $K - 1$ indices

```
function InitWorkingSet($\mathbf{x}$, $\nabla\mathcal{F}(\mathbf{x})$, $\rho$)
    $(\mathcal{S}_A,\, \mathcal{S}_I) \leftarrow (\{\varnothing\},\, \{\varnothing\})$
    $\mathbf{v} \in I\!R^n$
    for $i \leftarrow 1$ to $n$ do
        $w^{\star} \leftarrow \underset{w=r \text{ or } R}{\text{optimize}}\, (w - \mathbf{x}_i)\, \nabla\mathcal{F}(\mathbf{x})_i$
        $\mathbf{v}_i \leftarrow (w^{\star} - \mathbf{x}_i)\, \nabla\mathcal{F}(\mathbf{x})_i$
        if $\mathbf{v}_i \neq 0$ then $\mathcal{S}_A \leftarrow \mathcal{S}_A \cup i$
        else $\mathcal{S}_I \leftarrow \mathcal{S}_I \cup i$
    if $|\mathcal{S}_A| > K$ then
        $\mathcal{S} \leftarrow \{i_k\}_{k=1}^{K-1-\rho}$ s.t. $|\mathbf{v}_{i_k}|_{k<K-\rho} \geq |\mathbf{v}_{i_l}|_{l \geq K-\rho}$
        for $i \leftarrow 1$ to $\rho$ do $\mathcal{S} \leftarrow \mathcal{S} \cup \underset{i \in \mathcal{S}_A \setminus \mathcal{S}}{\text{random}}\ i$
    else
        $\mathcal{S} \leftarrow \mathcal{S}_A$
        for $i \leftarrow 1$ to $K - 1 - |\mathcal{S}_A|$ do $\mathcal{S} \leftarrow \mathcal{S} \cup \underset{i \in \mathcal{S}_I}{\text{random}}\ i$
    return $\mathcal{S}$
```

In other words, when the QP is maximized, we pick $\eta^{\star}_i$ such that $\eta^{\star}_i \nabla\mathcal{F}^{(t)}_i \geq 0$, and when the QP is minimized we pick $\eta^{\star}_i$ such that $\eta^{\star}_i \nabla\mathcal{F}^{(t)}_i \leq 0$. Then we define $\mathcal{S}_A^{(t)} = \{i_k\}$ such that $\eta^{\star}_{i_k} \nabla\mathcal{F}^{(t)}_{i_k} \neq 0$ and $\mathcal{S}_I^{(t)} = \{j_l\}$ such that $\eta^{\star}_{j_l} \nabla\mathcal{F}^{(t)}_{j_l} = 0$. The first $K - 1$ indices of $\mathcal{S}$ are chosen among $\mathcal{S}_A^{(t)}$ and the last index is chosen among $\mathcal{S}_A^{(t)} \cup \mathcal{S}_I^{(t)}$. When $|\mathcal{S}_A^{(t)}| \leq K$-1, all the indices of $\mathcal{S}_A^{(t)}$ are part of $\mathcal{S}$ and the remaining K-1-$|\mathcal{S}_A^{(t)}|$ indices are picked randomly from $\mathcal{S}_I^{(t)}$. When $|\mathcal{S}_A^{(t)}| > K$-1, we pick the best K-1-ρ indices $i_k \in \mathcal{S}_A^{(t)}$ corresponding to the largest $|\eta^{\star}_{i_k} \nabla\mathcal{F}^{(t)}_{i_k}|$. The other ρ indices are picked randomly in $\mathcal{S}_A^{(t)}$. While randomizing a portion of the indices for $\mathcal{S}_A^{(t)}$ may slow down the convergence rate a little bit, it prevents falling into poor local optimum. Typically we choose $\rho \ll |\mathcal{S}_A^{(t)}|$. Note that when $\mathbf{x}^{(0)}$ is sparse and when $\mathbf{x}^{\star}$ is expected to be sparse, $|\mathcal{S}_A^{(t)}|$ decreases quickly, reducing the search space for $\mathcal{S}$ dramatically. For the last index i_K, we run a search over all the unpicked indices and retain the one that offers the best optimizer $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S} \cup i_K}$ of Eqn. (4). Our heuristic generalizes [2], [30] as it not only introduces a random selection process in order to avoid poor local optimum, but also takes into account the potential displacement $\eta^{\star}$ along each gradient direction. For $\rho = 0$, $\eta = \pm 1$ and $K \leq 2$, our heuristic is equivalent to the one in [2], [30]. Our working set selection algorithm for the first K-1 indices is summarized in Alg. 1.
Eqn. (3) can also be reformulated to avoid the working set selection process, leading to the next lemma:

Lemma 2. *Eqn.* (3) *can be reformulated in the null space of the constraints so that $K \geq 1$.*

Proof. Let $|\mathcal{K}er(\mathbf{C})| = D$ and $\mathbf{U} \in I\!R^{n \times D}$ such that the columns $\{\mathbf{u}_i\}_{i=1}^{D}$ of $\mathbf{U}$ span $\mathcal{K}er(\mathbf{C})$. We can parametrize $\boldsymbol{\delta}\mathbf{x} = \mathbf{U}\boldsymbol{\alpha}$ since $\mathbf{C}\,\boldsymbol{\delta}\mathbf{x} = \mathbf{0} \Leftrightarrow \boldsymbol{\delta}\mathbf{x} \in \mathcal{K}er(\mathbf{C})$. Then Eqn. (3) becomes:

$$\begin{aligned} \boldsymbol{\alpha}^{\star} \leftarrow \underset{\boldsymbol{\alpha}}{\text{optimize}}\quad & \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{U}\boldsymbol{\alpha} + (\mathbf{A}\mathbf{x}^{(t)} + \mathbf{A}^{\mathsf{T}}\mathbf{x}^{(t)} + \mathbf{b})^{\mathsf{T}}\mathbf{U}\boldsymbol{\alpha} \\ \text{s.t.}\quad & r \leq \mathbf{x}^{(t)}_i + (\mathbf{U}\boldsymbol{\alpha})_i \leq R \quad \forall i \in \{1,\, n\} \end{aligned} \qquad (6)$$

Indeed $\forall \boldsymbol{\alpha} \in I\!R^D$, $\mathbf{CU}\boldsymbol{\alpha} = \mathbf{0}$, i.e. $\mathbf{CU} = \mathbf{0}$ by construction. From Lemma 1 $\mathbf{CU} = \mathbf{0} \Leftrightarrow \text{rank}(\mathbf{CU}) = 0 \Rightarrow K \geq 1$. $\square$

The downside of this reformulation is that we need to pay the space and computation price of processing $\mathbf{U}$ and $\mathbf{U}^\intercal\mathbf{AU}$, unless $\mathbf{A}$ is extremely sparse or D extremely low. One clear advantage is when $D = |\mathcal{K}er(\mathbf{C})| \leq 2$, optimizing Eqn. (6) is equivalent to optimizing a bounded D-dimensional conic equation, as demonstrated by Eqn. (13) and Eqn. (15) that have closed form solutions (see next two sections).

V. Solving the Minimal QP Problem

Many applications (min/max clique, cut, independent set, etc) have constraints with rank$(\mathbf{C}) \leq 1$. Therefore, in the following we will show how to solve the small QP problem of Eqn. (4) for special cases where rank$(\mathbf{C}) = 0$, 1 and for the general case of rank$(\mathbf{C}) \geq 2$. In the future, we will set $K = \text{rank}(\mathbf{C}) + 1$.

A. Special Case for rank($\boldsymbol{C}$) $= \mathbf{0}$, $\boldsymbol{K} = \mathbf{1}$

rank$(\mathbf{C}) = 0$ means the linear constraint $\mathbf{Cx} = \mathbf{d}$ is removed from Eqn. (2). Since $K = 1$, $\mathcal{S} = \{k\}$ and $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = \alpha \in I\!R$. At each iteration t Eqn. (4) becomes:

$$\begin{aligned}(k^\star,\, \alpha^\star) \leftarrow \underset{k,\,\alpha}{\text{optimize}} \quad & \alpha^2\mathbf{A}_{kk} + \alpha\,(\mathbf{Ax}^{(t)} + \mathbf{A}^\intercal\mathbf{x}^{(t)} + \mathbf{b})_k \\ \text{s.t.} \quad & \mathbf{x}_k^{(t)} + \alpha \in [r,\, R]\end{aligned} \tag{7}$$

We solve Eqn. (7) with respect to α for every index $k \in \{1, n\}$. For each fixed k, Eqn. (7) reduces to optimizing a bounded second degree equation of the form $\alpha^\star \leftarrow \text{optimize}\, \beta\alpha^2 + \gamma\alpha$, where alpha's bounds are defined as $\alpha_{\min} = r - \mathbf{x}_k^{(t)}$ and $\alpha_{\max} = R - \mathbf{x}_k^{(t)}$. The next section gives the closed form solution for $\alpha^\star$. At iteration t, the complexity of solving Eqn. (7) is linear.

B. Special Case for rank($\boldsymbol{C}$) $= \mathbf{1}$, $\boldsymbol{K} = \mathbf{2}$

With rank$(\mathbf{C}) = 1$ the linear constraints in Eqn. (2) reduces to $\mathbf{c}^T\mathbf{x} = \mathbf{d}$, with $\mathbf{c}$ and $\mathbf{d} \in I\!R^n$. $\mathcal{S} = \{k,\, l\}$ and $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = [\alpha_k,\, \alpha_l]^\intercal \in I\!R^2$. As described in Section IV, we first conduct a linear search to find k, then we perform a second linear search to find l, α_k and α_l. In order to find k we search the dimension that would lead to highest potential gradient magnitude along the dimension k, taking into account the bounds on $\mathbf{x}_k \in [r,\, R]$:

$$k^\star \leftarrow \underset{k \in \mathcal{S}_A^{(t)}}{\arg\max} \left\| \underset{w = r \text{ or } R}{\text{optimize}}\, (w - \mathbf{x}_k)(\mathbf{Ax} + \mathbf{A}^\intercal\mathbf{x} + \mathbf{b})_k \right\| \tag{8}$$

With a high probability we keep $k^\star$, and with a low probability we simply randomly pick $k \in \mathcal{S}_A^{(t)}$ such that $|\eta\nabla\mathcal{F}(\mathbf{x})_k| \neq 0$. We then perform a second linear search to find the second dimension l and the coefficients $[\alpha_k,\, \alpha_l]$:

$$\begin{aligned}(l^\star,\, \alpha_k^\star,\, \alpha_l^\star) \leftarrow \underset{l \neq k, \alpha_l, \alpha_k}{\text{optimize}}\; & \alpha_k^2\mathbf{A}_{kk} + \alpha_l^2\mathbf{A}_{ll} + \alpha_k\alpha_l(\mathbf{A}_{kl} + \mathbf{A}_{lk}) \\ & + \alpha_k(\mathbf{Ax}^{(t)} + \mathbf{A}^\intercal\mathbf{x}^{(t)} + \mathbf{b})_k + \alpha_l(\mathbf{Ax}^{(t)} + \mathbf{A}^\intercal\mathbf{x}^{(t)} + \mathbf{b})_l \\ \text{s.t.}\;\; \alpha_k\mathbf{c}_k + \alpha_l\mathbf{c}_l = 0,\; & \text{ with } \mathbf{x}_k^{(t)} + \alpha_k \text{ and } \mathbf{x}_l^{(t)} + \alpha_l \in [r,\, R]\end{aligned} \tag{9}$$

First let's assume both $\mathbf{c}_{k,\,l} \neq 0$. Solving the constraints gives:

$$\begin{aligned}\alpha_k = -\frac{\alpha_l\mathbf{c}_l}{\mathbf{c}_k} \quad & \text{and} \quad \alpha_{min_l} \leq \alpha_l \leq \alpha_{max_l} \\ \text{where} \quad \alpha_{min_l} &= \max\left(r - \mathbf{x}_l^{(t)},\, \frac{(\mathbf{x}_k^{(t)} - W)\mathbf{c}_k}{\mathbf{c}_l}\right) \\ \text{and} \quad \alpha_{max_l} &= \min\left(R - \mathbf{x}_l^{(t)},\, \frac{(\mathbf{x}_k^{(t)} - w)\mathbf{c}_k}{\mathbf{c}_l}\right)\end{aligned} \tag{10}$$

where $(w,\, W) = (r,\, R)$ if $\mathbf{c}_k\mathbf{c}_l > 0$ or $(w,\, W) = (R,\, r)$ if $\mathbf{c}_k\mathbf{c}_l < 0$. Then, solving Eqn. (9) reduces to optimizing a bounded second degree equation of the form:

$$\begin{aligned}(l^\star,\, \alpha_k^\star,\, \alpha_l^\star) \leftarrow \underset{l \neq k, \alpha_l}{\text{optimize}} \quad & \beta_l\alpha_l^2 + \gamma_l\alpha_l \\ \text{s.t.} \quad \alpha_l \in [\alpha_{\min_l},\, \alpha_{\max_l}] \quad & \text{and} \quad \alpha_k^\star = -\frac{\alpha_l^\star\mathbf{c}_l}{\mathbf{c}_k}\end{aligned} \tag{11}$$

Now let us assume that only one of the coefficients $(\mathbf{c}_k,\, \mathbf{c}_l)$ is null. We deduce from the linear constraint that $\mathbf{c}_k = 0 \Rightarrow \alpha_l = 0$ and $\mathbf{c}_l = 0 \Rightarrow \alpha_k = 0$. In both cases Eqn. (9) reduces again to optimizing a bounded second degree equation. Finally when $[\mathbf{c}_k,\, \mathbf{c}_l]^T = \mathbf{0}$ Eqn. (9) reduces to optimizing a bounded conic equation. The closed form solutions for both the bounded second degree and the bounded conic equations are shown in Section VI.

C. General Case for rank($\boldsymbol{C}$) $\geq \mathbf{2}$, $\boldsymbol{K} =$ rank($\boldsymbol{C}$) + 1

As described in Section IV, our algorithm first estimates the K-1 indices of $\mathcal{S}$ and next conducts a line search on the remaining dimension i_K while optimizing Eqn. (4) with respect to $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}}$. As a result, for every candidate index i_K, we simply need to optimize Eqn. (4) with respect to $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}}$. This ensures that both $i_K^\star$ and $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}^\star}^\star$ will be chosen as the best optimizer of Eqn. (4) at the end.

Lemma 3. *If $K =$ rank$(\boldsymbol{C}) + 1$ and rank$(\boldsymbol{C}_{\mathcal{S}}) = \kappa$ then the complexity of optimizing Eqn.* (4) *with respect to $\boldsymbol{\delta x}_{\mathcal{S}}$ is reduced to the complexity of (i) solving the linear system $\boldsymbol{C}_{\mathcal{S}}\boldsymbol{\delta x}_{\mathcal{S}} = 0$ and (ii) optimizing a D-dimensional bounded conic equation with $D =$ nullity$(\boldsymbol{C}_{\mathcal{S}}) =$ rank$(\boldsymbol{C}) - \kappa + 1$.*

Proof. $\mathbf{C}_{\mathcal{S}} \in I\!R^{m \times K}$ hence from Def. 2 nullity$(\mathbf{C}_{\mathcal{S}}) = K -$ rank$(\mathbf{C}_{\mathcal{S}}) =$ rank$(\mathbf{C}) - \kappa + 1$. Solving $\mathbf{C}_{\mathcal{S}}\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = 0$ is equivalent to finding the vector space $[\mathbf{u}_1,\, \ldots,\, \mathbf{u}_D]$ spanning $\mathcal{K}er(\mathbf{C}_{\mathcal{S}})$, with $D =$ nullity$(\mathbf{C}_{\mathcal{S}})$. We can then parametrize $\boldsymbol{\delta}\mathbf{x}_{\mathcal{S}} = \sum_{k=1}^{D}\alpha_k\mathbf{u}_k$ and solve Eqn. (4) with respect to $\boldsymbol{\alpha} = [\alpha_1,\, \ldots,\, \alpha_D] \in I\!R^D$:

$$\begin{aligned}\boldsymbol{\alpha}^\star \leftarrow \underset{\boldsymbol{\alpha}}{\text{optimize}} \sum_{k=1}^{D}\alpha_k^2\mathbf{A}_{i_ki_k} + \sum_{k=1,l=1}^{D}\alpha_k\alpha_l\mathbf{A}_{i_ki_l} + \sum_{k=1}^{D}\alpha_k\nabla\mathcal{F}_{i_k}^{(t)} \\ \text{s.t. } \alpha_k \in [\alpha_{\min_k},\, \alpha_{\max_k}] \text{ and } \nabla\mathcal{F}^{(t)} = \mathbf{Ax}^{(t)} + \mathbf{A}^\intercal\mathbf{x}^{(t)} + \mathbf{b}\end{aligned} \tag{12}$$

Eqn. (12) is a D-dimensional bounded conic equation. $\square$

For $D = 1$, 2 we provide a closed form solution in Section VI. When $D > 2$ we need to optimize iteratively the $D-$dimensional bounded conic equation, i.e. we need to solve a sub-QP problem of dimension D. Note that we can always randomly pick two linear combinations of vectors in $\mathcal{K}er(\mathbf{C}_\mathcal{S})$ and solve a bounded 2D conic equation, which is the equivalent of one optimization step in the random projection algorithm in [34]. Finally finding the vector space spanning $\mathcal{K}er(\mathbf{C}_\mathcal{S})$ is done analytically via QR decomposition or SVD of $\mathbf{C}_\mathcal{S}$. Optimizing the $D-$dimensional bounded conic equation can be achieved with time complexity $O(D)$ where the time complexity for the QR decomposition and Gram-Schmidt basis completion process is $O(m^3 + m(K-m)^2)$ or for the SVD is $O(m^2K + mK^2 + K^3)$. Solving the small QP problem is summarized in Alg. 2 and our final SMO algorithm is described in Alg. 3.

Algorithm 2 Solve the small QP problem.

```
1: function SolveSmallQP(x, $\nabla\mathcal{F}^{(t)}$, A, C, $\mathcal{S}$)
2:     Find $\{\mathbf{u}_k\} \in \mathcal{K}er(\mathbf{C}_\mathcal{S})$        ▷ Using for example QR on $\mathbf{C}_\mathcal{S}$
3:     Parametrize $\boldsymbol{\delta}\mathbf{x}_\mathcal{S} = \sum \mathbf{u}_k \boldsymbol{\alpha}_{\mathcal{S}k}$
4:     Solve Eqn. (4) w.r.t. the $\{\boldsymbol{\alpha}_{\mathcal{S}k}\}$
5:     return ($\boldsymbol{\delta}\mathbf{x}^\star_\mathcal{S}$)
```

Note that when rank$(\mathbf{C}) \geq n-2$, the sub-QP problem can be solved analytically. More generally when rank$(\mathbf{C}_\mathcal{S}) \geq K-2$, i.e. when rank$(\mathbf{C}) \leq$ rank$(\mathbf{C}_\mathcal{S})+1$, the bounded conic (or second degree) equation has a closed form solution described in Section VI. For $D > 2$, iterative approaches to optimize the D-dimensional bounded conic equation are more efficient then closed form solutions, and one can simply reuse the same SMO algorithm to solve the sub-QP problem.

VI. Optimizing bounded second degree and bounded 2D conic equations

The bounded second degree equation from the previous section has the following form:

$$\alpha^\star \leftarrow \underset{\alpha}{\text{optimize}} \quad \beta\alpha^2 + \gamma\alpha \quad \text{s.t.} \quad \alpha \in [\alpha_{\min},\, \alpha_{\max}] \tag{13}$$

where $\alpha^\star$ has a closed form solution depending on the sign of β and whether the optimization problem Eqn. (2) is a maximization or minimization problem. When solving the maximization problem $\mathbf{x}^\star \leftarrow \arg\max \mathcal{F}(\mathbf{x})$, $\alpha^\star$ has the closed form solution:

$$\alpha^\star \leftarrow \begin{cases} -\frac{\gamma}{2\beta} & \text{if } -\frac{\gamma}{2\beta} \in [\alpha_{\min},\, \alpha_{\max}] \text{ and } \beta < 0 \\ \underset{\substack{\alpha=\alpha_{\min} \\ \text{or } \alpha_{\max}}}{\arg\max} \beta\alpha^2 + \gamma\alpha & \text{otherwise} \end{cases} \tag{14}$$

When solving the minimization $\mathbf{x}^\star \leftarrow \arg\min \mathcal{F}(\mathbf{x})$, $\alpha^\star$ has the same closed form solution as Eqn. (14), except that $\arg\max$ becomes $\arg\min$ and $\beta < 0$ becomes $\beta > 0$. In some cases, for rank$(\mathbf{C}_\mathcal{S}) = K-2$, the QP minimal problem cannot be reduced to a bounded second degree equation, but instead to a bounded conic equation of the form:

$$\begin{aligned} \boldsymbol{\alpha}^\star = [\boldsymbol{\alpha}_1^\star,\, \boldsymbol{\alpha}_2^\star]^T \leftarrow \underset{\boldsymbol{\alpha}_1,\, \boldsymbol{\alpha}_2}{\text{optimize}} \quad & \boldsymbol{\beta}_{11}\boldsymbol{\alpha}_1^2 + \boldsymbol{\beta}_{22}\boldsymbol{\alpha}_2^2 + \\ & (\boldsymbol{\beta}_{12} + \boldsymbol{\beta}_{21})\boldsymbol{\alpha}_1\boldsymbol{\alpha}_2 + \boldsymbol{\gamma}_1\boldsymbol{\alpha}_1 + \boldsymbol{\gamma}_2\boldsymbol{\alpha}_2 \\ \text{s.t.} \quad \boldsymbol{\alpha}_1 \in [\alpha_{\min_1},\, \alpha_{\max_1}] & \text{ and } \boldsymbol{\alpha}_2 \in [\alpha_{\min_2},\, \alpha_{\max_2}] \end{aligned} \tag{15}$$

In order to solve Eqn. (15), we first compute the optimum on the boundaries, i.e we fix $\boldsymbol{\alpha}_1 = \alpha_{\min_1}$ and optimize the second degree equation with respect to $\boldsymbol{\alpha}_2$. We then repeat the procedure by fixing $\boldsymbol{\alpha}_1 = \alpha_{\max_1}$, $\boldsymbol{\alpha}_2 = \alpha_{\min_2}$ and $\boldsymbol{\alpha}_2 = \alpha_{\max_2}$. Finally, if $(\boldsymbol{\beta}_{12} + \boldsymbol{\beta}_{21})^2 - 4\boldsymbol{\beta}_{11}\boldsymbol{\beta}_{22} < 0$ we check if there's an optimum inside the bounds by solving the 2×2 linear system: $2\boldsymbol{\beta}\boldsymbol{\alpha} = -\boldsymbol{\gamma}$. Note that $(\boldsymbol{\beta}_{12} + \boldsymbol{\beta}_{21})^2 - 4\boldsymbol{\beta}_{11}\boldsymbol{\beta}_{22} < 0 \Leftrightarrow 0 \leq (\boldsymbol{\beta}_{12} - \boldsymbol{\beta}_{21})^2 < \det(\boldsymbol{\beta})$, which guarantees $\boldsymbol{\beta}$ is invertible. Our final SMO algorithm is described in Alg. 3.

Algorithm 3 General SMO (G-SMO)

```
1: Initialize:
2:     $K = \text{rank}(\mathbf{C}) + 1$
3:     $\mathbf{x}^{(0)}$ such that $\mathbf{C}\mathbf{x}^{(0)} = \mathbf{d}$ and $\forall i \in \{1,\, n\},\, \mathbf{x}_i^{(0)} \in [r,\, R]$
4:     $\nabla\mathcal{F}^{(0)} \leftarrow (\mathbf{A} + \mathbf{A}^\intercal)\mathbf{x}^{(0)} + \mathbf{b}$
5: Main loop:
6:     for $t \leftarrow 1$ to $t_{max}$ do
7:         $\mathcal{S} \leftarrow$ InitWorkingSet($\mathbf{x}^{(t)}$, $\nabla\mathcal{F}^{(t)}$, $\rho$)
8:         $\delta\mathcal{F}^\star \leftarrow 0$
9:         for $j \in \{1,\, n\} \setminus \mathcal{S}$ do    ▷ Search for the last index of $\mathcal{S}$
10:            $\mathcal{S} \leftarrow \mathcal{S} \cup j$
11:            $\boldsymbol{\delta}\mathbf{x}^\star_\mathcal{S} \leftarrow$ SolveSmallQP($\mathbf{x}^{(t)}$, $\nabla\mathcal{F}^{(t)}$, A, C, $\mathcal{S}$)
12:            $\delta\mathcal{F} \leftarrow \left\|\boldsymbol{\delta}\mathbf{x}_\mathcal{S}{}^\intercal \mathbf{A}_{\mathcal{S}\mathcal{S}} \boldsymbol{\delta}\mathbf{x}_\mathcal{S} + \nabla\mathcal{F}_\mathcal{S}^{(t)\intercal} \boldsymbol{\delta}\mathbf{x}_\mathcal{S}\right\|$
13:            if $\delta\mathcal{F} > \delta\mathcal{F}^\star$ then $j^\star, \delta\mathcal{F}^\star, \boldsymbol{\delta}\mathbf{x}^\star_{\mathcal{S}^\star} \leftarrow j, \delta\mathcal{F}, \boldsymbol{\delta}\mathbf{x}^\star_\mathcal{S}$
14:            $\mathcal{S} \leftarrow \mathcal{S} \setminus j$
15:        $\mathcal{S}^\star \leftarrow \mathcal{S} \cup j^\star$
16:        $\mathbf{x}^{(t+1)}_{\mathcal{S}^\star} \leftarrow \mathbf{x}^{(t)}_{\mathcal{S}^\star} + \boldsymbol{\delta}\mathbf{x}^\star_{\mathcal{S}^\star}$
17:        $\nabla\mathcal{F}^{(t+1)} \leftarrow \nabla\mathcal{F}^{(t)} + (\mathbf{A} + \mathbf{A}^\intercal + diag(\mathbf{b}))_{\mathcal{S}^\star}\, \boldsymbol{\delta}\mathbf{x}^\star_{\mathcal{S}^\star}$
18:        if $\delta\mathcal{F}^\star < \varepsilon$ then break
19: return $\mathbf{x}^\star$
```

VII. Applications and Results

In this section, we apply our algorithm to several problems. For all experiments, the code was written in C++ with SSE optimization and ran on an 2.8 GHz Intel Core i7, with 16 GB 1600 MHz DDR3 RAM.

A. Single Doc. Summarization as a Sparse Max Cut

In order to generate summaries of a single document with the most N relevant sentences, we model the problem as a sparse maximum cut problem [35], [36]:

$$\begin{aligned} \mathbf{x}^\star \leftarrow \underset{\mathbf{x}}{\arg\max} \quad & \mathcal{F}(\mathbf{x}) = (\mathbf{1} - \mathbf{x})^\intercal(\mathbf{A} - \mathbf{D})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{1}^\intercal\mathbf{x} = N \text{ and } \mathbf{x} \in [0,\, 1]^n \end{aligned} \tag{16}$$

where $\mathbf{A}_{ij}$ is the similarity between sentences i and j, with $|\mathbf{A}_{ij}| \leq 1$. $\mathbf{D}$ is a diagonal matrix such that $\mathbf{D}_{ii} \geq \max_j \mathbf{A}_{ij}$. Hager et al. [37] proved that introducing $\mathbf{D}$ yields to a proper binary solution $\mathbf{x}$ for the aforementioned maximum cut problem. Also note that when $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ the QP problem becomes equivalent to solving the minimum dominant set formulation of [38]. In the following we will report results for $\mathbf{D}_{ii} = \max_j \mathbf{A}_{ij}$ (MCUT), for $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ (MDOM) and for $\mathbf{D}_{ii} = 1$ (CONST1). We used the following similarity metric between sentences:

$$\mathbf{A}_{ij} = \frac{\sum_k \mathbf{hi}_k \wedge \mathbf{hj}_k}{\sum_k \mathbf{hi}_k \vee \mathbf{hj}_k} \text{ with } \mathbf{hi}_k = \begin{cases} 1, \text{ if sentence i contains} \\ \quad \text{the } k^{th} \text{ word of the} \\ \quad \text{document dictionary} \\ 0, \text{ otherwise} \end{cases} \tag{17}$$

In order to evaluate our algorithm adapted to perform summarization, we used the English sub-corpus of Concisus Corpus of Event Summaries [39], which contains 78 event reports and respective summaries, distributed across three different types of events: aviation accidents, earthquakes, and train accidents. To evaluate the quality of the summary, we used standard ROUGE [40], namely ROUGE-1, which is the most widely used evaluation measure for this scenario. We generate 3 sentence summaries, commonly found in online news web sites, like Google News. We compared our algorithms against LexRank [41] and Centrality [42] for a better understanding of the improvements. We also measured Max ROUGE-1 which was obtained by selecting the highest ROUGE-1 value among all possible combination of three sentences.

Table I outlines the result of our algorithms and the baselines on the Concisus dataset. Method CONST1 outperforms the LexRank and Centrality baselines, as well as the other two methods MDOM and MCUT.

Alg.	LexRank	Centrality	MDOM	MCUT	CONST1
ROUGE-1	0.428	0.443	0.482	0.489	**0.506**

TABLE I
ROUGE-1 RESULTS ON THE CONCISUS DATASET. MAX ROUGE-1: 0.646.

	# feat.	train. size	test. size
adu.	123	$1605 \rightarrow 32561$	$30956 \rightarrow 16281$
web.	300	$2477 \rightarrow 49749$	$47272 \rightarrow 14951$
cod.	8	59535	271617
spl.	60	1000	2175

TABLE II
DATASET SPECIFICATIONS. THE ADU. AND WEB. DATASETS ARE PARTITIONED IN INCREASING TRAINING SET SIZE AND DECREASING TESTING SET SIZE. THE SYMBOL "$a \rightarrow b$" INDICATES THE SIZE RANGE. THE SAME TERMINOLOGY WILL BE USED FOR TABLE III.

	adu.	web.	cod.	spl.
acc. LIBSVM (%)	79.5 ± 0.5	$\mathbf{97.7 \pm 0.3}$	66.7	83.6
acc. ours (%)	$\mathbf{80.4 \pm 1}$	97.3 ± 0.04	**85.1**	**84**
#SV LIBSVM	200	∼200	200	200
#SV ours	$\mathbf{137.8 \pm 67.9}$	$\mathbf{110 \pm 18.9}$	**39**	**197**
t.t. LIBSVM (ms)	$28.8 \rightarrow 683.8$	$92.9 \rightarrow 2343.7$	176.6	22.9
t.t. ours (ms)	$\mathbf{7.82 \rightarrow 142.3}$	$\mathbf{6.32 \rightarrow 182.9}$	**70.1**	**8.3**

TABLE III
ACCURACY (ACC.), NUMBER OF SUPPORT VECTORS (#SV) AND TRAINING TIME (T.T.) FOR THE BEST PARAMETER γ ON SVM BINARY CLASS DATASETS FOR LIBSVM AND OUR APPROACH.

B. Support Vector Machine

We wanted to make sure that our algorithm was still performing comparably to the LIBSVM for the problem of SVM, even though we extended SMO to a more general class of problems. In order to evaluate our algorithm we used the datasets "adult" (adu.), "webpage" (web.), "cod-rna" (cod.), and "splice" (spl.) from [43]. The adult dataset is composed of nine partitions: a1a $\rightarrow$ a9a and the webpage dataset is composed of height partitions: w1a $\rightarrow$ w8a. For both datasets, each partition starts with a small training set and a large testing one, and ends with a large training set and a small testing one. We run our experiments on each partition. Implementation-wise, we used the standard LIBSVM code wrapped in the OpenCv library with the RBF kernel $K_{RBF}(\mathbf{x}^i, \mathbf{x}^j) = e^{-\gamma \|\mathbf{x}^i - \mathbf{x}^j\|^2}$. For our approach we used $\bar{K}(\mathbf{x}^i, \mathbf{x}^j) = 1 - K_{RBF}(\mathbf{x}^i, \mathbf{x}^j)$, $\varepsilon = 10^{-5}$. We initialize our system with $\leq$10 random support vectors. We also used the same grid search ($\gamma \in [0, 1]$, $\gamma_{\text{incr.}} = 10^{-5}$) for both methods for fair comparison. Note that our implementation did not account for further optimizations including caching and shrinking, other heuristics based on the support vector margins or decision constraints [2]. Table II shows the baseline on the aforementioned datasets and Table III shows the results of our algorithm. While our approach provides slightly better accuracy, our solution is also sparser in terms of numbers of support vectors. Hence our approach only takes fewer iterations to converge, leading to a training time smaller than with the standard LIBSVM approach.

C. Toy Experiments for rank($\mathbf{C}$) > 2

In order to test our approach for rank($\mathbf{C}$) > 2, we used an extended version of the maximum cut problem described in Eqn. (16). The new QP problem is formulated as follows:

$$\begin{aligned} \mathbf{x}^\star \leftarrow \arg\max_{\mathbf{x}} \quad & \mathcal{F}(\mathbf{x}) = (\mathbf{1} - \mathbf{x})^\mathsf{T}(\mathbf{A} - \mathbf{I})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{C}\mathbf{x} = \mathbf{d},\ \mathbf{I} = \text{identity},\ \mathbf{x} \in [0, 1]^n \\ & \mathbf{A} \in \{0, 1\}^{n \times n},\ \mathbf{C} \in \{-1, 1\}^{m \times n} \\ \text{where} \quad & \mathbf{C}_1^\mathsf{T} = \mathbf{1} \text{ and } \mathbf{d}_1 = N \text{ and } \forall i > 1 \\ & \mathbf{C}_{ij} \in \{0, 1\}^n,\ (\mathbf{C}\,\mathbf{1})_i = 0 \text{ and } \mathbf{d}_i = 0 \end{aligned} \tag{18}$$

$\mathbf{A}$ is randomly initialized with a level of sparsity of 70%, i.e. only 30% of the values of $\mathbf{A}$ are 1s. We fix rank($\mathbf{C}$) $= m$, and to guarantee that $\mathbf{C}$ has m independent columns, we build $\mathbf{C}$ as follows:

$$\begin{aligned} \mathbf{C} &= \left[\begin{array}{c|c|c} \multicolumn{3}{c}{\mathbf{1}^\mathsf{T}} \\ \hline 2\mathbf{I} - \mathbf{1}\mathbf{1}^\mathsf{T} & \mathbf{1} & \mathbf{R} \end{array}\right] \qquad \mathbf{R} \in \{-1, 1\}^{m-1 \times m-2} \\ &= \left[\begin{array}{cccc|c|ccc} 1 & \multicolumn{6}{c}{\dots\dots\dots\dots\dots\dots\dots} & 1 \\ \hline 1 & -1 & \dots & -1 & 1 & -1 & 1 & \dots \\ -1 & 1 & \dots & -1 & 1 & 1 & -1 & \dots \\ -1 & -1 & \ddots & \vdots & \vdots & 1 & 1 & \dots \\ -1 & -1 & \dots & 1 & 1 & 1 & 1 & \dots \end{array}\right] \end{aligned} \tag{19}$$

Here the first m columns of $\mathbf{C}$ are guaranteed to be independent from the $m-1 \times m-1$ sub-matrix $2\mathbf{I}$–$\mathbf{1}\mathbf{1}^\mathsf{T}$. It is also clear that the m^{th} column is also independent from the first m-1's. In order to guarantee that $(\mathbf{C}\,\mathbf{1})_{i>1} = 0$, we define $\mathbf{R}$ such that $(\mathbf{R}\,\mathbf{1})_i = m - 4$, i.e. each row of $\mathbf{R}$ has $(m - 2 + \frac{n-m}{2})$ 1's and $(2 + \frac{n-m}{2})$ -1's. Since we also want to guarantee that the constraints $\mathbf{C}\mathbf{x} = \mathbf{d}$ can be respected in the box space $\mathbf{x} \in [0, 1]^n$, for $k < m$ we choose $\mathbf{R}_k = -(2\mathbf{I} - \mathbf{1}\mathbf{1}^\mathsf{T})_k$, and for $k \geq m$ we simply set $\mathbf{R}_k$ to be a random column of $2\mathbf{I} - \mathbf{1}\mathbf{1}^\mathsf{T}$ and $\mathbf{R}_{k+1} = -\mathbf{R}_k$. Finally we randomly swap columns and rows of $\mathbf{C}$ to minimize the biased introduced by the sequential construction of $\mathbf{C}$. The identity matrix $\mathbf{I}$ is introduced to push the values of $\mathbf{x}$ to be binary. The additional $m-1$ constraints of $\mathbf{C}$ can be viewed at compatibility constraints enforced on the solution, i.e. while $\mathbf{A}$ represents the node connectivity in the graph, $\mathbf{C}^{\intercal}_{2<i<m}$ represent sets of hyper-graph edges of node affinities for the final solution.

We set $N=5$ and test two additional variants of Alg. 3 for $m=4$ and $m=n-2$ with $n=100, 1000, 10000$. For each $\{m,\, n\}$ combination we run 100 instances of the QP problem with different $\mathbf{A}$ and $\mathbf{C}$. The algorithm variant *"random working set"* (RWS-SMO) is replacing the working set selection by picking randomly unique indices for $\mathcal{S}^{\star}$. The variant *"null space reformulation"* (NSR-SMO) is based on the problem reformulation Eqn. (6) of Lemma 2. Note that we do not compare with simplex or interior point methods because naive use of general LP solvers would be computationally too expensive for large n and m. Table IV and V show the results of the different algorithms for $m=4$ and $m=n-2$. The RWS-SMO is suboptimal as it picks dimensions that

	$n=100$		$n=1000$		$n=10000$	
	$\mathcal{F}^{\star}_{\text{avg}}$	Time	$\mathcal{F}^{\star}_{\text{avg}}$	Time	$\mathcal{F}^{\star}_{\text{avg}}$	Time
RWS-SMO	110.3	7.14	1301.1	30.2	13832.4	1324.8
NSR-SMO	128.2	5.71	1321.9	441.2	**14366.1**	$1.62\ 10^{5}$
G-SMO	**132.1**	**1.42**	**1389.5**	**9.35**	14233.7	**121.7**

TABLE IV
AVERAGE SCORE $\mathcal{F}^{\star}_{\text{AVG}}$ AND PROC. TIME (MS) FOR $m=4$.

	$n=100$		$n=1000$		$n=10000$	
	$\mathcal{F}^{\star}_{\text{avg}}$	Time	$\mathcal{F}^{\star}_{\text{avg}}$	Time	$\mathcal{F}^{\star}_{\text{avg}}$	Time
RWS-SMO	101.2	32.8	1252.5	$8.41\ 10^{5}$	–	$\gg 1.10^{8}$
NSR-SMO	**115.1**	8.12	1289.0	**684.8**	**12249.9**	$\mathbf{4.08\ 10^{5}}$
G-SMO	113.5	**7.90**	**1317.1**	4580.4	11975.4	$9.52\ 10^{6}$

TABLE V
AVERAGE SCORE $\mathcal{F}^{\star}_{\text{AVG}}$ AND TIME (MS) FOR $m=n-2$.

don't necessarily need to be optimized, hence it takes more iterations to converge and the local optimum is worst than the other two approaches, showing the importance of the working set selection. For $m=4$, the G-SMO algorithm is more efficient than its variants as it quickly recursively decomposes the small QP problem into smaller ones until a closed form solution can be used since $m \ll n$. For $m=n-2$ the NSR-SMO variant is more efficient as it directly reduces to a bounded 2D conic equation.

VIII. Discussion

In this section we want to tackle three aspects of our approach: (i) generalization of our approach to non-convex problems, (ii) convergence of Algorithm 3 and (iii) scalability of our approach. First, we do not make any assumption regarding the nature of the problem to solve, i.e. we do not take advantage of any particular structure for the matrix in the objective function (symmetric, positive/negative, semi-definite, etc), nor for the constraint system. If the problem is non-convex, the optimum is only guaranteed to be local, as any other approaches trying to solve NP problems. Other technics, like simulated annealing, or additional knowledge on the initial solution can certainly improve the local optimum, but this is not the scope of the paper. However we do guarantee that Algorithm 3 converge to an optimum. Indeed, step 13 enforces that the objective function keeps being optimized and step 18 exits the iterative process when the objective function does not change any more. Finally our approach inherits from the low memory cost of the original SMO as it solves iteratively the minimal QP problems. The memory required at each iteration is linearly proportional to the rank of the constraints, making our approach scalable to large dataset when the constraints have low rank. When the constraints have very high rank ($\geq$ n-1) we also showed that we can reformulate the problem directly in the null space of the constraints.

IX. Conclusion

In this research, we extended SMO to a more general form of QP problem than the initial SVM one, and used it to derive novel results in several fields of machine learning including document summarization and sparse SVM. First, we proved the conditions under which SMO can be used. Second, we established strategies and bounds in order to reduce the general QP problem to optimizing a sequence of smallest possible QP problems. Third, we show an 18% improvement on single-document summarization without using any supervised information. This result will enable substantial improvements in other summarization tasks, including multi-document summarization. Finally we showed that we can reformulate QP problems using graph theory problem transformations directly. Our sparse maximum clique formulation of SVM improves the accuracy of the original formulation by minimizing the model memory footprint and training time. In future research, we plan to explore additional application including schedule and task planning, Part-of-Speech tagging, image/video segmentation, and object tracking.

References

[1] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Advances in Kernel Methods - Support Vector Learning, Tech. Rep., 1998.

[2] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order information for training support vector machines," *J. Mach. Learn. Res.*, vol. 6, pp. 1889–1918, Dec. 2005.

[3] N. Deo, *Graph Theory with Applications to Engineering and Computer Science (Prentice Hall Series in Automatic Computation)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1974.

[4] R. P. Singh and Vandana, "Article: Application of graph theory in computer science and engineering," *International Journal of Computer Applications*, vol. 104, no. 1, pp. 10–13, October 2014.

[5] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE TPAMI*, vol. 22, no. 8, 2000.

[6] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 29, p. 2007, 2007.

[7] W. Brendel and S. Todorovic, "Segmentation as maximum-weight independent set." in *NIPS*, 2010.

[8] M. Pavan and M. Pelillo, "Dominant sets and pairwise clustering," *IEEE TPAMI*, 2007.

[9] E. Zemene and M. Pelillo, "Interactive image segmentation using constrained dominant sets," *CoRR*, vol. abs/1608.00641, 2016.

[10] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *IJCV*, 2004.

[11] B. Peng, L. Zhang, and D. Zhang, "A survey of graph theoretical approaches to image segmentation," *Pattern Recogn.*, vol. 46, no. 3, pp. 1020–1038, Mar. 2013.

[12] W. Brendel, M. R. Amer, and S. Todorovic, "Multiobject tracking as maximum weight independent set," in *CVPR*, 2011, pp. 1273–1280.

[13] A. R. Zamir, A. Dehghan, and M. Shah, "Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs," ser. ECCV'12, 2012, pp. 343–356.

[14] A. Dehghan, S. Modiri Assari, and M. Shah, "Gmmcp tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking," in *CVPR*, June 2015.

[15] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *JMLR'01*.

[16] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast svm training on very large data sets," *J. Mach. Learn. Res.*, vol. 6, pp. 363–392, Dec. 2005.

[17] W. W. Hager and J. T. Hungerford, "Continuous quadratic programming formulations of optimization problems on graphs," *European Journal of Operational Research*, vol. 240, no. 2, pp. 328–337, 2015.

[18] L. Liu, T. G. Dietterich, N. Li, and Z. Zhou, "Transductive optimization of top k precision," *CoRR*, vol. abs/1510.05976, 2015.

[19] S. G. Vadlamudi, S. Sengupta, S. Kambhampati, M. Taguinod, Z. Zhao, A. Doupé, and G. Ahn, "Moving target defense for web applications using bayesian stackelberg games," *CoRR*, vol. abs/1602.07024, 2016.

[20] X. Wu, D. Sheldon, and S. Zilberstein, "Optimizing resilience in large scale networks," in *Proceedings of the Thirtieth Conference on Artificial Intelligence*, 2016.

[21] Q. Zhou, W. Chen, S. Song, J. R. Gardner, K. Q. Weinberger, and Y. Chen, "A reduction of the elastic net to support vector machines with an application to gpu computing," in *Proceedings of AAAI*, 2015, pp. 3210–3216.

[22] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou, "Ranking with recursive neural networks and its application to multi-document summarization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15, 2015, pp. 2153–2159.

[23] Z. Li and J. Chen, "Superpixel segmentation using linear spectral clustering," in *CVPR'15*.

[24] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, pp. 99–131, Jan. 2005.

[25] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT workshop*. ACM, 1992, pp. 144–152.

[26] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Workshop of NNSP*. IEEE, 1997, pp. 276–285.

[27] G. Fung and O. L. Mangasarian, "Finite newton method for lagrangian support vector machine classification," *Neurocomputing*, vol. 55, pp. 39–55, 2003.

[28] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR Comp. Mathem. and Mathematical Physics*, 1967.

[29] Y. Censor and A. Lent, "An iterative row-action method for interval convex programming," *JOTA*, 1981.

[30] S. R. Bul and I. M. Bomze, "Infection and immunization: A new class of evolutionary game dynamics." *Games and Economic Behavior*, vol. 71, no. 1, pp. 193–211, 2011.

[31] P. Chen, R. Fan, and C. Lin, "A study on smo-type decomposition methods for support vector machines," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 893–908, 2006.

[32] S. V. N. Vishwanathan, Z. sun, N. Ampornpunt, and M. Varma, "Multiple kernel learning and the smo algorithm." in *NIPS*, 2010, pp. 2361–2369.

[33] S. Boyd and L. Vandenberghe, *Convex Optimization*. NY, USA: Cambridge Univ. Press, 2004.

[34] S. Paul, C. Boutsidis, M. Magdon-Ismail, and P. Drineas, "Random projections for linear support vector machines," *ACM Trans. Knowl. Discov. Data*, vol. 8, no. 4, pp. 22:1–22:25, Aug. 2014.

[35] H. Lin and J. Bilmes, "Multi-document summarization via budgeted maximization of submodular functions," in *NAACL*, 2010, pp. 912–920.

[36] ——, "A class of submodular functions for document summarization," in *ACL*, 2011.

[37] W. W. Hager, D. T. Phan, and H. Zhang, "An exact algorithm for graph partitioning," *CoRR'09*.

[38] C. Shen and T. Li, "Multi-document summarization via the minimum dominating set," in *COLING*, 2010.

[39] H. Saggion and S. Szasz, "The concisus corpus of event summaries." in *LREC*, 2012.

[40] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *ACL workshop*, 2004.

[41] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

[42] R. Ribeiro, L. Marujo, D. Martins de Matos, J. P. Neto, A. Gershman, and J. Carbonell, "Self reinforcement for important passage retrieval," in *SIGIR*. ACM, 2013, pp. 845–848.

[43] C.-C. Chang and C.-J. Lin, "www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html," LIBSVM Data: Binary Classification.