Optimization Methods and Software

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/goms20

On solving the densest *k*-subgraph problem on large graphs

Renata Sotirov

To cite this article: Renata Sotirov (2020) On solving the densest *k*-subgraph problem on large graphs, Optimization Methods and Software, 35:6, 1160-1178, DOI: 10.1080/10556788.2019.1595620

To link to this article: https://doi.org/10.1080/10556788.2019.1595620

OPTIMIZATION METHODS & SOFTWARE
2020, VOL. 35, NO. 6, 1160–1178
https://doi.org/10.1080/10556788.2019.1595620

Taylor & Francis
Taylor & Francis Group

On solving the densest *k*-subgraph problem on large graphs

Renata Sotirov

Department of Econometrics and OR, Tilburg University, Tilburg, Netherlands

ABSTRACT
The densest *k*-subgraph problem is the problem of finding a *k*-vertex subgraph of a graph with the maximum number of edges. In order to solve large instances of the densest *k*-subgraph problem, we introduce two algorithms that are based on the random coordinate descent approach. Although it is common use to update at most two random coordinates simultaneously in each iteration of an algorithm, our algorithms may simultaneously update many coordinates. We show the benefit of updating more than two coordinates simultaneously for solving the densest *k*-subgraph problem, and solve large problem instances with up to 2^{15} vertices.

ARTICLE HISTORY
Received 4 June 2018
Accepted 12 March 2019

KEYWORDS
Densest *k*-subgraph problem; random coordinate descent algorithm; large graphs

2010 MATHEMATICS SUBJECT CLASSIFICATIONS
90C06; 90C30; 90C26

1. Introduction

The densest *k*-subgraph (D*k*S) problem is the problem of finding a subgraph of the given graph with exactly *k* vertices such that the number of edges in the subgraph is maximal. The densest *k*-subgraph problem is known in the literature under various names, including the heaviest unweighted subgraph problem [35], the *k*-cluster problem [15], or the *k*-cardinality subgraph problem [14]. The densest *k*-subgraph problem can be seen as a special case of the maximum *k*-dispersion problem [47]. The maximum *k*-dispersion problem is the problem of finding *k* vertices in a graph that maximize a function of the distances between the chosen vertices. The D*k*S problem can also be seen as a special case of the heaviest *k*-subgraph problem, which is the problem of finding a subgraph with *k* vertices that maximizes the sum of the edge weights in the subgraph.

The D*k*S problem is known to be NP-hard. In [21] it is proven that the problem is NP-hard for graphs whose maximum degree is equal to three. The densest *k*-subgraph problem is NP-hard even for very restricted classes of graphs, such as bipartite and chordal graphs [15], or planar graphs [30]. However, it is trivial on trees. The D*k*S problem is solvable in polynomial time on graphs whose maximum degree is equal to two, as well as on cographs, split graphs, and *k*-trees, see [15].

There are many applications of the problem. The densest *k*-subgraph problem plays a role in analysing web graphs and different social networks. Namely, one of the main challenges for web search engines is the detection of link spams, see Henzinger *et al.* [28]. Link spams are websites that are linked to each other in order to manipulate the search engine

CONTACT Renata Sotirov r.sotirov@uvt.nl

rankings. Many of the dense subgraphs in web graphs are link spams. Gibson *et al.* [23] propose an algorithm that extracts dense subgraphs in huge graphs in order to identify link spams. Angel *et al.* [5] analyse social networks to identify real-time stories by searching for dense subgraphs of the given size. The heaviest k-subgraph problem can be used to create interest groups of people. For example, when organizing an opening party where participants should be as similar as possible, or to analyse political vote data, see [50]. The heaviest k-subgraph may be used to find teams of employees with the highest collaborative compatibility, see [22].

We next list problems that are related to the DkS problem. The following two versions of the densest k-subgraph problem are introduced in [4]: the densest at-least-k-subgraph problem and the densest at-most-k-subgraph problem. The densest at-least-k-subgraph (resp. the densest at-most-k-subgraph) problem is the problem of finding an induced subgraph of highest average degree with at least (resp. at most) k vertices. Andersen and Chellapilla [4] present an efficient 1/3-approximation algorithm for the densest at-least-k-subgraph problem. There are no efficient approximation algorithms for the densest at-most-k-subgraph problem. The problem of finding a subgraph of maximum node weights with exactly k edges is considered in [25]. The sparsest k-subgraph problem finds the subgraph with k vertices and the minimum number of edges, see [11].

Outline and main results. This paper is structured as follows. Section 2 presents an integer programming formulation of the problem and lists various solving approaches from the literature. In Section 3 we provide an overview of recently introduced methods for solving large scale optimization problems. We present two new algorithms for solving the DkS problem in Section 4. Our algorithms are applied to the relaxation of the DkS problem, see (6) Our first algorithm considers a quadratic optimization subproblem with linear constraints, and the second one a linear programming optimization subproblem. The main difference between the here presented algorithms and those in the literature is that we allow updating more than two random coordinates simultaneously in each iteration of our algorithms. We show here that for an appropriate number of simultaneous updates, our second algorithm converges to an integer solution vector (!). This convergence is not proven with theoretical convergence analysis, but only empirically observed.

Our extensive numerical results show that we find densest subgraphs in large graphs in short time, see Section 5. For example, we find a densest subgraph with 25 vertices in a graph with 23,133 vertices and 93,497 edges in less than 4 min. Since the densest subgraph is a clique in this case, we know that we found an optimal subgraph. Exact approaches for finding densest k-subgraphs can not cope with graphs that have more than 160 vertices. On the other hand, the best heuristic approaches are tested on random instances with at most 3000 vertices. We test our algorithms on real-world data and on randomly generated data with up to 32,768 vertices.

2. The problem formulation and solution approaches

The densest k-subgraph problem can be formulated as a quadratic optimization problem with binary variables. Let $G = (V, E)$ be an undirected graph with vertex set V, $|V| = n$, and edge set E, $|E| = m$. Let k be a positive integer between 3 and $n-2$, and x_i a binary variable that obtains value one if vertex i is in the densest k-subgraph and zero otherwise. We denote by A the adjacency matrix of G.

The densest k-subgraph problem can be formulated as follows:

$$\begin{aligned} \max \quad & x^{\mathrm{T}} A x \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = k \\ & x_i \in \{0,1\}, \quad \forall\, i \in \{1,\ldots,n\}. \end{aligned} \tag{1}$$

In the sequel, we list approaches that are used for solving the DkS. Billionnet [8] derived four different mixed-integer linear programming formulations for the heaviest k-subgraph problem, and three different mixed-integer linear programming formulations for the densest k-subgraph problem. Numerical results in [8] show that the quality of a formulation is related to the density of an instance. In [9], the authors solve instances of the densest k-subgraph problem by reformulating the non-convex quadratic problem (1) into an equivalent problem with a convex objective function. Such reformulation requires solving an associated semidefinite programming problem. The reformulated problem is then solved by using a branch-and-bound algorithm. This approach is tested on random graphs with at most 100 vertices. Numerical results show that the proposed convexification approach improves efficiency of the branch and bound algorithm. However, solving the related semidefinite program may be costly.

Malick and Roupin [40] solve instances of the DkS problem to optimality using semidefinite programming. Namely, they solve a semidefinite programming problem in each node of a branch and bound tree. The largest solved instances of the DkS has 120 vertices. Krislock *et al.* [37] report solving hard instances of the DkS problem with up to 160 vertices by using a semidefinite branch and bound algorithm. Semidefinite programming relaxations are also used in the design of approximation algorithms for the DkS, see e.g. [20,53]. One can find an overview of SDP relaxations for the densest k-subgraph problem in [48]. The above mentioned results show that it is extremely difficult to find a densest k-subgraph in a graph that has more than 160 vertices by using exact approaches.

A number of recent results have focused on recovering planted k-subgraphs by using convex relaxation techniques, see e.g. [1,2]. Ames and Vavasis [2] show that the maximum clique in a graph consisting of a single large clique can be identified from the minimum nuclear norm solution of a particular system of linear inequalities. Ames [1] establishes analogous recovery guarantees for a convex relaxation of the planted clique problem that is robust to noise. For a survey on the topic see Li *et al.* [39].

In 2001, Feige *et al.* [19] provide an approximation algorithm for the DkS problem with approximation ratio of $n^{\delta-\epsilon}$ for some small ϵ. In [7], it is presented an approximation algorithm that for every $\epsilon > 0$ approximates the DkS problem within a ratio of $n^{1/4+\epsilon}$ in $n^{O(1/\epsilon)}$ time. The most recent results on the superpolynomial approximation algorithms for the DkS one can find in [12]. In [32], Khot proves that there does not exist a polynomial time approximation scheme (PTAS) for the densest k-subgraph problem in general graphs. However, there exist polynomial time approximation schemes for a few special problem cases. Arora *et al.* [6] provide a PTAS for the DkS problem on dense instances. Nonner [45] drives a PTAS for interval graphs.

Different heuristic methods are tested for solving the densest k-subgraph problem. Kincaid [33] uses simulated annealing and tabu search heuristics to solve the DkS problem. His

results show that the tabu search algorithm performs better than the simulated annealing algorithm for solving the densest k-subgraph problem. In [41], Macambira implements tabu search heuristics for the heaviest k-subgraph problem. Although the tabu search algorithm from [41] does not perform diversification, it outperforms the greedy randomized adaptive search procedure. A variable neighborhood search (VNS) heuristics for the heaviest subgraph problem and graphs up to 3000 vertices is implemented by Brimberg *et al.* [13]. Their results show that the VNS outperforms the tabu search heuristic and multi-start local search heuristics in solving the DkS. The VNS performs extremely well on sparse graphs. Running times needed to find the best solutions for instances with 3000 vertices is about 425 s. A heuristic based on a two-step filtering approach is used to extract dense web communities in Dourisboure *et al.* [16].

3. Overview of methods for large scale optimization

Nesterov [42] introduced constrained and unconstrained versions of an efficient method for solving convex huge-scale optimization problems. Followed by that paper, appeared different versions of coordinate descent methods for large scale convex optimization, see e.g. [43,44,49]. In this paper, we propose two variants of the random coordinate descent method to solve the DkS. In this section we provide a brief overview of algorithms from the literature, and describe those that are relevant to our work in more details.

The random coordinate descent method (RCDM) from [42], is a method for solving unconstrained problems with convex objective. The RCDM performs in every iteration of the algorithm a random coordinate index selection by using a random counter. The random counter generates numbers according to a distribution that is based on the coordinatewise Lipschitz constants. The uniform coordinate descent method (UCDM) from [42] is developed for solving constrained problems with convex objective. The method uses the uniform distribution to determine random coordinates. In the UCDM, each coordinate update is based on a solution of an optimization subproblem. The optimization subproblem considers constraints of the original problem, and takes care that the new point is in the vicinity of the previous one. The methods introduced in [42] turn to be efficient for solving huge scale convex optimization problems.

There exist several extensions of the random coordinate descent method and uniform coordinate descent method from [42]. For example, the random block coordinate descent method for linearly constrained optimization by Necoara *et al.* [44]. This method is introduced for solving problems with a separable convex objective function and one linear constraint. Richtárik and Takáč [49] extend results from [42] to composite optimization. In particular, they introduce randomized block-coordinate descent methods for minimizing composite functions. Another recently developed method for solving large-scale optimization problems is a subgradient method by Nesterov [43]. The approach from [43] is suitable for optimization problems with sparse subgradients. The above mentioned methods are tested on large or huge scale convex problems such as the Google's PageRank problem, the PageRank problem, image processing, estimation in sensor networks or distributed control, l_1-regularized least squares problems.

On the other hand, there are very few results on solving large-scale non-convex problems. Patrascu and Necoara [46] derive random coordinate descent algorithms for large

scale structured non-convex optimization problems, and test them on sparse instances of the eigenvalue complementarity problem.

Before we outline the UCDM from [42] and the 2-random coordinate descent algorithm from [46], we introduce the notation. Consider the space $\mathbb{R}^N$, and its decomposition on n subspaces where $N = \sum_{i=1}^{n} n_i$. We denote a block decomposition of $N \times N$ identity matrix by $I_N = (U_1, \ldots, U_n) \in R^{N \times N}$, where $U_i \in \mathbb{R}^{N \times n_i}$ $(i = 1, \ldots, n)$. Thus, for $x = (x^{(1)}, \ldots, x^{(n)}) \in \mathbb{R}^N$ we have

$$x = \sum_{i=1}^{n} U_i x^{(i)}, \tag{2}$$

where $x^{(i)} \in \mathbb{R}^{n_i}$ for $i = 1, \ldots, n$. Note that similar notation is used in the related literature, see e.g. [42].

Let us now describe the UCDM from [42]. Consider a function $f(x)$ that is convex and differentiable on a closed convex set $Q \subseteq \mathbb{R}^N$. Assume that the gradient of f is coordinatewise Lipschitz continuous with constants L_i $(i = 1, \ldots, n)$ where

$$\|\nabla_i f(x + U_i h_i) - \nabla_i f(x)\| \leq L_i \|h_i\| \quad h_i \in \mathbb{R}^{n_i}, \quad i = 1, \ldots, n, \quad x \in \mathbb{R}^N, \tag{3}$$

$\nabla_i f(x)$ denotes the partial gradient of $f(x)$ in $x^{(i)}$, i.e.

$$\nabla_i f(x) = U_i^{\mathrm{T}} \nabla f(x) \in \mathbb{R}^{n_i}, \quad x \in \mathbb{R}^N,$$

and $\|\cdot\|$ denotes the Euclidean norm.

Now, the constrained optimization problem considered in [42] is:

$$\min_{x \in Q} f(x),$$

where $Q = \bigotimes_{i=1}^{n} Q_i$ and the sets $Q_i \subseteq \mathbb{R}^{n_i}$ $(i = 1, \ldots, n)$ are closed and convex. The ith $(i = 1, \ldots, n)$ constrained coordinate update from [42] is:

$$V_i(x) = x + U_i^{\mathrm{T}}(u^{(i)}(x) - x^{(i)}), \tag{4}$$

where and $u^{(i)}(x)$ is the solution of the following optimization subproblem:

$$u^{(i)}(x) = \arg \min_{u^{(i)} \in Q_i} \left[\langle \nabla_i f(x), u^{(i)} - x^{(i)} \rangle + \frac{L_i}{2} \|u^{(i)} - x^{(i)}\|^2 \right]. \tag{5}$$

Here $\langle \cdot, \cdot \rangle$ denotes a vector product. The uniform coordinate descent method [42] chooses a random number i from the discrete uniform distribution and updates $x^{(i)}$ in every iteration. In particular, see Algorithm 1.

The improvement in each step of the UCDM is as follows:

$$f(x) - f(V_i(x)) \geq \frac{L_i}{2} \|u^{(i)}(x) - x^{(i)}\|^2.$$

Patrascu and Necoara [46] introduce random coordinate descent algorithms for large-scale structured non-convex optimization problems. They consider unconstrained and

Algorithm 1 Algorithm UCDM [42]

Require: A feasible initial solution x_0.
  $k \leftarrow 0$
  loop
    Choose randomly i_k by uniform distribution on $\{1, \ldots, n\}$.
    Update $x_{k+1} = V_{i_k}(x_k)$, by using (4) and (5).
    $k \leftarrow k + 1$
  end loop

linearly constrained problems with a non-convex and composite objective function. In particular, in [46] it is considered the following linearly constrained optimization problem:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & g(x) + \text{ł}(x) \\ \text{s.t.} \quad & a^{\mathrm{T}} x = b, \end{aligned}$$

where $a \in \mathbb{R}^N$ is a non-zero vector, $b \in \mathbb{R}$, g is a smooth function, and ł is a convex, separable, non-smooth function. Further, the function g has 2-block coordinate Lipshitz continuous gradient, i.e. there exist constants $L_{ij} > 0$ such that

$$\|\nabla_{ij}\, g(x + U_i h_i + U_j h_j) - \nabla_{ij}\, g(x)\| \leq L_{ij} \|h_{ij}\|$$

for all $h_{ij} = [h_i^{\mathrm{T}}, h_j^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{n_i + n_j}$, $x \in \mathbb{R}^N$ and $i, j = 1, \ldots, n$. For given a feasible initial point x_0, that is $a^{\mathrm{T}} x_0 = b$, the 2-random coordinate descent algorithm from [46] is presented as Algorithm 2.

Algorithm 2 Algorithm 2-RCD [46]

Require: A feasible initial solution x_0.
  $k \leftarrow 0$
  loop
    Choose randomly 2 block coordinates (i_k, j_k) with probability p_{i_k, j_k}.
    Update $x_{k+1} = x_k + U_{i_k} d_{i_k} + U_{j_k} d_{j_k}$.
    $k \leftarrow k + 1$
  end loop

Here, directions $d_{i_k, j_k} = [d_{i_k}^{\mathrm{T}}, d_{j_k}^{\mathrm{T}}]^{\mathrm{T}}$ are obtained from the following optimization subproblem

$$\begin{aligned} d_{i_k, j_k} = \arg\min_{h_{i_k, j_k}} \quad & g(x_k) + \langle \nabla_{i_k, j_k}\, g(x_k), h_{i_k, j_k} \rangle + \frac{L_{i_k, j_k}}{2} ||h_{i_k, j_k}||^2 + \text{ł}(x_k + h_{i_k, j_k}) \\ \text{s.t.} \quad & a_{i_k}^{\mathrm{T}} h_{i_k} + a_{j_k}^{\mathrm{T}} h_{j_k} = 0. \end{aligned}$$

In [46], the authors prove asymptotic convergence of the sequence generated by 2-RCD to stationary points. The authors suggest updating only two block coordinates simultaneously in each iteration of the algorithm. Thus, the 2-RCD algorithm updates only two

coordinates simultaneously when $n = N$ i.e. for the scalar case. In the following section we consider updating more than two coordinates in each iteration of our algorithms. Our numerical results show that updating more than two (block) coordinates simultaneously results with a very efficient algorithm.

4. Two new algorithms for solving the D*k*S

We present here two new algorithms for solving large instances of the D*k*S. The algorithms are tailored for solving the relaxation of the D*k*S, but can be adjusted for solving any non-convex problem with a linear constraint. Our algorithms can be seen as extensions of the UCDM and the 2-RCD algorithms, see Section 3. While our first algorithm converges to a real feasible point, our second algorithm provides an integer feasible point in most of the cases, see Section 5. A version of the here presented second algorithm was studied in a master thesis by van der Doef [52].

Let us first consider the following relaxation of (1):

$$\begin{aligned} \max \quad & x^{\mathrm{T}} A x \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = k \\ & 0 \le x_i \le 1, \quad \forall i \in \{1, \dots, N\}. \end{aligned} \tag{6}$$

Recall that the constrained coordinate update in the UCDM algorithm considers one block coordinate, while the coordinate update in the 2-RCD algorithm updates two block coordinates in each iteration. To solve the D*k*S we update several coordinates simultaneously, in each iteration of our algorithms.

We set $f(x) = x^{\mathrm{T}} A x$ and suppose that $n = N$, see (2). In each step of our coordinate descent algorithms we update $q \ge 2$ coordinates. Let J_i, $|J_i| = q$, be the set of random coordinates that are updated simultaneously in step i. Then, our q-random coordinate constrained update in ith iteration is as follows:

$$W^i(x)_j = \begin{cases} u_j^i(x) & \text{if } j \in J_i \\ x_j & \text{otherwise} \end{cases} \qquad j = 1, \dots, n, \tag{7}$$

where $u^i(x) \in \mathbb{R}^q$ is the solution of a concave optimization subproblem. In particular,

$$\begin{aligned} u^i(x) \quad = \quad \arg\max_{u^i} \quad & \sum_{j \in J_i} \nabla_j f(x)(u_j^i - x_j) - \sum_{j \in J_i} \frac{L_j}{2}(u_j^i - x_j)^2 \\ \text{s.t.} \quad & \sum_{j \in J_i} u_j^i = k - \sum_{j \notin J_i} x_j \\ & 0 \le u_j^i \le 1 \quad \forall j \in J_i. \end{aligned} \tag{8}$$

Here, L_j is defined as in (3). Note that the q-random coordinate constrained update in ith iteration can be also written as:

$$W^i(x) = x + \sum_{j \in J_i} U_j^{\mathrm{T}} (u_j^i(x) - x_j).$$

It is a well known result that convex quadratic problems are polynomially solvable, see [36]. Kozlov *et al.* [36] reported $O(n^4L)$ algorithm for convex quadratic problems, where n is the number of variables and L is the size of the problem. Later papers present algorithms that have complexities of $O(n^3L)$ arithmetic operations, see e.g. [26,34].

Now, we are ready to show our first algorithm. For a fixed q ($2 \leq q \leq n$) the q-random coordinate constrained algorithm q-RCC1 is presented as Algorithm 3.

Algorithm 3 Algorithm q-RCC1

Require: A feasible initial solution x_0.
  $k \leftarrow 0$
  loop
    Determine J_k: choose q coordinates randomly by uniform distribution on $\{1, \ldots, n\}$.
    Update $x_{k+1} = W^k(x_k)$ by using (7) and (8).
    $k \leftarrow k + 1$
  end loop

Note that one can solve (8) efficiently with a convex quadratic programming solver.

Since the optimization problem (6) is non-convex, the algorithm q-RCC1 can stuck in a local optimum. Therefore, we also allow restarting of the algorithm from a new feasible starting point. The algorithm q-RCC1 uses several stopping criteria. For details on restarting and stopping criteria, see Section 5. Numerical results show that q-RCC1 converges to a local optimum of the relaxation problem (6). However, we are interested in solving the integer programming problem (1). Therefore, the subproblem of our next algorithm considers only a linear approximation of the non-convex objective function. In particular, our second algorithm solves the following subproblem in order to find a q-random coordinate constrained update in ith iteration:

$$\begin{aligned} u^i(x) \quad = \quad \arg\max_{u^i} \quad & \sum_{j \in J_i} \nabla_j f(x)(u^i_j - x_j) \\ \text{s.t.} \quad & \sum_{j \in J_i} u^i_j = k - \sum_{j \notin J_i} x_j \\ & 0 \leq u^i_j \leq 1 \quad \forall j \in J_i. \end{aligned} \tag{9}$$

Thus, to obtain the q-random coordinate constrained update from (9), we need to solve a linear programming problem. In 1979, Khachiyan [31] proved that linear programming is polynomially solvable. Karamarker's well known projective algorithm, see [29], solves linear programs in $O(n^4L)$ operations, where n is the number of variables in a standard-form problem with integer data of bit size L. Many subsequent papers have reported $O(n^3L)$ algorithms for linear programming. Anstreicher [3] shows that the complexity to solve linear programming problems can be reduced to $O([n^3/\ln n]L)$.

Our q-random coordinate constrained algorithm q-RCC2 for a fixed q ($2 \leq q \leq n$) is given as Algorithm 4.

Our numerical results show that the algorithm q-RCC2 converges to an integer point for sufficiently large q. Once the algorithm finds an integer point, all points in successive iterations are also integer. Moreover, the successive integer vectors might be in the vicinity

Algorithm 4 Algorithm q-RCC2

Require: A feasible initial solution x_0.
$k \leftarrow 0$
loop
Determine J_k: choose q coordinates randomly by uniform distribution on $\{1, \ldots, n\}$.
$x_{k+1} = W^k(x_k)$ by using (7) and (9).
$k \leftarrow k + 1$
end loop

of the first found integer solution. Therefore, the q-RCC2 algorithm could end up again in the first found integer solution. To prevent cycling, we stop the q-RCC2 algorithm once the first integer solution is found, or we restart the algorithm from a new feasible point.

5. Numerical results

In this section we present numerical results on solving the DkS problem by using our two algorithms, i.e. q-RCC1 and q-RCC2. Numerical results are performed on an Intel Xeon, E5-1620, 3.70 GHz with 32 GB memory. To compute (8) (resp. (9) we use Cplex 12.6 QP (resp. Cplex 12.6 LP) solver.

We test our algorithms on random graphs and several graphs from the literature. In particular, we consider the following graphs:

(1) The Erdös-Rényi graph: Each edge in a graph is generated independently of other edges with probability $p \in (0, 1]$. For any given p, a graph formulated in the described way is known as the Erdös–Rényi random graph $G_p(n)$. The Erdös-Rényi graph was introduced by Erdös and Rényi in 1959, see [17,18].
(2) The Erdös-Rényi graph with a planted subgraph: In the Erdös–Rényi random graph $G_p(n)$ we plant a complete subgraph with k vertices. We denote the resulted graph by $P_p^k(n)$. Random graphs with planted subgraphs are used also in Tsourakakis *et al.* [51]. We compare our results with their heuristic results.
(3) Instances for the DkS from the literature:
   - We consider instances that are available from the following webpage: http://cedric.cnam.fr/~lamberta/Library/k-cluster.html. Those instances are used as test instances for the densest k-subgraph problem in [8–10,37]. The sizes of instances are $n = 40,80,100,120,140,160$, and densities $d = 25, 50, 75\%$. For a given number of vertices n and a density d an unweighted graph is randomly generated. In all above mentioned papers, the parameter k has following values: $n/2$, $n/4$ and $3n/4$. Therefore, we also use the same values for k in our experiments.
   - Brimberg *et al.* [13] generated test instances for the heaviest k-subgraph problem. One can download those instances from the following page http://www.mi.sanu.ac.rs/~nenad/hsp/. There are 177 instances in total, for details see [13].
(4) Real-world data: We consider data from the following two different sources.
   - We test our algorithms on several graphs from 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering. In particular, we consider Jazz graph and Email graph. These two graphs are derived from two

different networks and then symmetrized, as explained on the DIMACS webpage: https://www.cc.gatech.edu/dimacs10/archive/clustering.shtml

- We take graphs from the following webpage: snap.stanford.edu. In particular, we use undirected graphs from the collaboration networks database. A collaboration network represents scientific collaborations between authors of papers in a specific field. A graph from the database is represented by an adjacency matrix whose element on position (i, j) equals one if author i co-authored a paper with author j. The largest here considered graph from the snap.stanford.edu database has 23,133 vertices.

We present below settings of our algorithms:

- We tested two different types of initial feasible points. The first type of an initial point is a random point in which all coordinates have values between zero and one, and all sum up to k. The second type of an initial point is the vector whose all coordinates equal to k/n. Our numerical results show that the quality of a solution computed by our algorithms does not depend on a starting point. Therefore, our algorithms start with a randomly generated feasible point unless indicated differently. Namely, it is costly to use the first type of the starting point when $n > 2^{13}$, and therefore we use the second type of initial point for larger n.
- We implement several stopping criteria. Both algorithms q-RCC1 and q-RCC2 stop after a pre-specified number of iterations is reached, unless any other stoping criteria is satisfied. We list the remaining stoping criteria below.
  - Stop when the first integer solution is found. Our tests show that after the first integer solution is found, q-RCC2 might cycle i.e. end up in the same integer point after a certain number of iterations. This happens since the algorithm computes integer points the vicinity of the first found integer solution. Our tests show that it is better to stop the algorithm when the first integer solution is found, instead of letting it run till eventually cycling appears and then stop. Namely, the latter requires more computational effort, but does not necessarily result in a significant improvement of the objective value. On the other hand, q-RCC1 does not converge to an integer solution, in general. However, for large q the algorithm q-RCC1 might also provide an integer solution. In the case that q-RCC1 finds an integer point, we stop the algorithm.
  - Stop q-RCC1 if the difference in two consecutive objective values is less than a pre-specified tolerance. We use here $\epsilon = 1\text{e}{-7}$ as the tolerance. This criteria is not implemented in q-RCC2 since the algorithm tends to find faster an integer value than to satisfy this criterion.
- Restarting of the algorithms. We sometimes perform restarting of the algorithms q-RCC1 and q-RCC2 for a given number of times and after one of the stopping criteria from above is reached. In each new run, we restart the algorithm by using one of the previously described starting points. It is going to be clear from the context if we performed restarting of the algorithm.

It might happen that in an iteration of our algorithm, the objective value decreases and then in the next iterations keeps improving. This happens since the objective is

non-convex. Extensive test shows that there is no harm in accepting non-improving moves, since the algorithms recover fast. We tested our algorithms also when only improving moves are accepted, and concluded that there is no benefit of doing this.

Let us now present computational results. We test our two algorithms on various instances, and present lower bounds for the problem (6) obtained from the limit point returned by the algorithms.

Tests on the Erdös-Rényi graphs.

We first show performance of our algorithms on $G_{0.5}(2^{10})$ for different number of simultaneous updates q and different number of iterations.

Table 1 presents bounds computed by the algorithm q-RCC1 for one graph only with 1024 vertices and $k = 30$. Here, we do not restart q-RCC1. The initial point in all runs have coordinates k/n with objective value 622.54. The table reads as follows. In the first row we specify q. Rows indicated by bnd. provide bounds that are computed in seconds, given in the first row below that one. Finally, rows indicated by iter. specify the number of iterations needed to compute bounds listed in the two rows above that one.

All computations in Table 1 terminated after the algorithm performed a pre-specified number of iterations. The results in Table 1 show that the quality of bounds improve and corresponding computational times increase along with the number of iterations. The table also shows that for $q = 2$ there is a small improvement in the bound even after 10,000 iterations. Note also that for large number of updates i.e. $q = 750$ there is no significant improvement in the bound value when the number of iterations increases. Table 1 also indicates that a good strategy for computing bounds is to take q that is between 10% and 20% of the number of vertices in the graph.

Table 2 presents bounds computed by q-RCC2 for the same graph used in Table 1. Since the algorithm q-RCC2 with $q > 2$ terminates in most of the cases due to the stopping criteria 'the first integer solution is found', we present results obtained by averaging 20 bounds computed after 20 times restarting the algorithm with the same starting point. We present

Table 1. q-RCC1 for $G_{0.5}(1024)$: bounds, running times (s) and iterations.

q	2	50	100	200	500	750
bnd.	623.12	653.15	686.54	738.260	808.38	816.86
time	1.01	1.02	1.28	2.20	4.77	7.51
iter.	500	500	500	500	500	500
bnd.	623.67	682.96	741.23	795.44	825.81	837.91
time	1.95	2.10	2.56	9.56	9.56	15.18
iter.	1000	1000	1000	1000	1000	1000
bnd.	628.12	799.33	824.27	831.99	836.96	833.33
time	9.54	10.33	13.32	23.66	49.27	77.55
iter.	5000	5000	5000	5000	5000	5000
bnd.	634.59	819.93	837.06	842.81	837.99	840.83
time	16.55	21.02	27.35	48.50	102.74	154.94
iter.	10,000	10,000	10,000	10,000	10,000	10,000

Table 2. q-RCC2 for $G_{0.5}(1024)$: bounds, running times (s) and iterations.

q	2	50	100	200	500	750
bnd.	636.15	825.23	828.66	824.20	790.70	731.30
time	0.643	1.12	1.07	0.64	0.26	0.20
iter.	500	492.05	255.70	70.90	13.9	6.85

average of 20 runs for each q. The initial point in all runs is the vector with coordinates k/n. Table 2 shows that the average of 20 bounds is the best for 100 simultaneous updates. Note that for $q \geq 200$ the computational time significantly drops, but the quality of bounds deteriorate. Among all computed bounds the best integer value is 840. That value is obtained for $q = 50$ and for $q = 100$.

Tests on graphs with planted subgraphs.

We plant complete subgraphs in random graphs because we know the optimal value of the problem. This enables us to evaluate the performance of our algorithms. Note that heuristic approaches [13,51] report results for graphs with up to 3000 vertices.

Table 3 summarizes outcomes of our two algorithms on graphs with planted subgraphs and 4096 vertices. In particular, we consider the Erdös-Rényi graphs $G_{0.3}(2^{12})$ whose planted complete subgraphs have 100 vertices. Note that the optimal value of the densest 100-subgraph problem on the described graph is 9,900 with high probability. We run each algorithm with different q on 30 different $P_{0.3}^{100}(4096)$ graphs. In particular, we run 1000 iterations of q-RCC1 and 1000 iterations of q-RCC2 for each $q \in \{2, 400, 800, 2000\}$ and each graph. In the row denoted by q-RCC1 (resp. q-RCC2) we list the best obtained bound among 30 values for the given q, as well as the computational time in seconds needed to compute that bound.

An interesting result is that the algorithm q-RCC1 with $q = 2000$ computes the value 9899.99 for 28 different graphs. Coordinates of the solution vectors in those 28 cases differ at most for 1e–5 from the value 0 or 1. If we let run q-RCC1 with $q = 2000$ for 10,000 iterations the best obtained result is 9899.99996. Here, values of coordinates in the solution vector are within an error of 1e–6 from 0 or 1. It takes 1215 s to perform 10,000 iterations.

The algorithm q-RCC2 finds the value 9900 in 11, 8 and 13 cases for $q = 400$, $q = 800$, $q = 2000$, respectively. Table 3 reports the shortest computational time required to compute 9900 by q-RCC2 among all computations. The longest time needed to obtain 9900 by q-RCC2 and $q = 800$ (resp. $q = 2000$) is 35.03 s (resp. 72.64 s). Finally, q-RCC2 computes the weakest bound for $q = 2000$. The results in Table 3 show that the algorithm q-RCC2 performs better than q-RCC1 for all q. However, q-RCC2 can stop fast in a weak bound. On the other hand, q-RCC1 improves slowly and steadily.

Let us now consider a similar experiment as the previous one, see Table 4. In particular, in the Erdös–Rényi graph $G_{0.3}(2^{12})$ we plant a complete subgraph with 800 vertices, which results in $P_{0.3}^{800}(4096)$. Note that the optimal value of the densest 800-subgraph problem on the described graph is 639,200 with high probability. We run 1000 iterations of each of

Table 3. Bounds and running times (s) for $P_{0.3}^{100}(4096)$.

q	2	400	800	2000
q-RCC1	6185.23 (2.08)	6908.02 (24.63)	9872.83 (46.30)	9899.99 (121.53)
q-RCC2	6240.01 (0.43)	9900 (5.83)	9900 (2.42)	9900 (1.26)

Table 4. Bounds and running times (s) for $P_{0.3}^{800}(4096)$.

q	2	400	800	2000
q-RCC1	215,748.73 (1.97)	639,199.99 (25.05)	639,199.99 (46.48)	639,199.99 (121.07)
q-RCC2	228,805.97 (0.46)	639,200 (2.61)	639,200 (2.33)	639,200 (0.91)

the algorithms for 30 different graphs and for $q = 2,400,800,2000$. The algorithm q-RCC2 finds the value 639,200 in 28, 28 and 27 cases for $q = 400$, $q = 800$, $q = 2000$, respectively. The results in Table 4 indicate that the algorithm q-RCC2 finds faster and more frequently cliques with 800 vertices than cliques with 100 vertices.

Finally, in a similar experiment with 2000-planted subgraph problem, the optimal value is computed by q-RCC2 in 23, 28 and 27 cases for $q = 400$, $q = 800$, $q = 2000$, respectively.

We did also extensive tests on $G_{0.2}(2^{13})$ with planted cliques on 500 vertices, i.e. $P_{0.3}^{500}(2^{13})$. It is interesting to note that for those graphs and 800 simultaneous updates, q-RCC2 always finds the planted subgraph between 10 and 55 s.

Finally, we experiment with random graphs as in [51]. We plant a complete graph with 30 vertices in $G_p(3000)$ with $p \in \{0.008, 0.1, 0.5\}$. The algorithm q-RCC2 with $q = 100$ finds the clique within 6 s in $P_{0.008}^{30}(3000)$. The algorithm q-RCC2 with $q = 150$ finds the clique within 50 s in $P_{0.01}^{30}(3000)$. There are no computational times reported in [51]. However, in [51], the authors report that all considered algorithms find the clique in a graph $G_{0.008}(3000)$, and only one algorithm can find the clique in $G_{0.01}(3000)$. On the other hand, no algorithms from [51] could find the clique in $G_{0.5}(3000)$. We also couldn't find the clique in $G_{0.5}(3000)$, even after several restarting of the algorithm.

Tests on instances from the literature.

We consider instances from http://cedric.cnam.fr/ ∼lamberta/Library/k-cluster.html. Those instances are also used as test instances for the DkS in [8–10,37], see also http://www-lipn.univ-paris13.fr/BiqCrunch/results. We summarize the outcome of our computational experiments below.

For each instance with $n = 40$, any given density and any $k = 10,20,30$, our algorithm q-RCC2 finds an optimal solution within 0.1 s. In q-RCC1 we implement additional stopping criteria, that is to stop the algorithm when the objective value differs from the optimal objective value for less than 0.0001. The algorithm q-RCC1 provides such bounds within 2 s. We allow 1000 iterations per round in both algorithms. Solutions are mostly found in the first round of the algorithms. We test both algorithms for $q = 4$ and $q = 8$ and notice similar performance of the algorithms for both values of q.

For each instance with $n = 80$, any given density and any $k = 20,40,60$, the algorithm 16-RCC2 finds an optimal solution within 0.2 s, while 8-RCC2 needs at most 0.4 s. The algorithm 16-RCC1 performs better than 8-RCC1 and requires at most 13 s to obtain a bound that is close to the optimal solution. In most of the cases, 16-RCC1 finds an optimal solution in less than 4 s. Here, we use the same additional stopping criteria as for instances with 40 vertices. We allow 2000 iterations per round.

For each instance with $n = 100$, any given densities, and any $k = 25,50,75$ the algorithm 10-RCC2 finds an optimal solution in less than 1 s. There are several instances for which the algorithm runs up to 3 s. We set 3000 for the maximal number of iterations in one round. This enables 20-RCC1 to converge to an optimal solution of a given instance in at most 16 s.

For instances with $n = 120,140,160$ we tested only the algorithm q-RCC2. We take for q the value that is equal to 20% of the number of vertices in the given instance, and allow 3000 iterations per round. For most of the instances q-RCC2 finds optimal solutions within 4 s. For the instances `kcluster160-050-40-1.dat`, `kcluster160-050-40-5.dat`, `kcluster160-075-40-2.dat` and `kcluster160-075-80-4.dat` we needed to change the value of q in order to find optimal

solutions. In particular, we set q to be 15% of the number of vertices and found optimal solutions within 25 s.

Brimberg *et al.* [13] provide extensive computational experiments on solving the heaviest k-subgraph problem by using several heuristic approaches. In particular, they compare performances of the following heuristics: two greedy constructive heuristics (drop and add), two versions of variable neighbourhood search (VNS) heuristics (basic VNS and skewed VNS), two tabu search heuristics (TS1 and TS2) and two multi-start heuristics (MLS1 and MLS2). The results in [13] show that VNS heuristic preforms the best over other heuristics. On the other hand TS1 has the worst performance among tested approaches.

Here, we test the algorithm q-RCC2 on the same set of instances as in [13]. We compare our results with the VNS and TS1 heuristics that use random initial starting points, see Table 3 in [13]. In Table 5 we report the average $ deviation

$$\%\ \text{deviation} = \frac{\text{best value} - \text{rcc2}}{\text{best value}} \cdot 100,$$

where 'best value' denotes the best known solution reported in [13], and 'rcc2' denotes our bound. We also report average running time obtained by q-RCC2, see the last column in Table 5. To solve instances we set $q = 100$ and alow restarting the algorithm 100 times. For instance with 1000 nodes we set 7000 for the maximum number of iterations per round, while for instances with 3000 nodes we set 10,000 iterations per round.

The results in Table 5 show that our algorithm is performing better than TS1 and worse than VNS. Note that the average % deviation of our algorithm is within 2%. The algorithms from [13] are specialized for solving the heaviest k-subgraph problem, while we use the best settings for the DkS.

Tests on real-world graphs.

Jazz graph represents jazz musicians network related to $n = 198$ musicians, see [24]. There are $m = 2742$ edges in the graph, which represent the network of jazz musicians. It is known that this graph contains a clique with 30 vertices, see [51]. The algorithm q-RCC2 with $q = 30$ finds the clique after 2 times restarting the algorithm, which takes in total 0.07 s. If we use $q = 10$, then the algorithm finds the optimal clique after 10 times restarting the algorithm. For $q = 2$ the algorithm q-RCC2 fails to find a clique even after restarting the algorithm 100 times.

Table 5. Summary results for all three types of the heaviest k-subgraph problem.

Type	n	k	% deviation q-RCC2	VNS	TS1	time q-RCC2
I sparse	1000	300	0.90	0.15	1.64	113.05
I sparse	1000	400	0.55	0.10	0.99	103.64
I sparse	1000	500	0.24	0.03	0.55	89.90
I dense	1000	300	0.24	0.04	0.49	112.91
I dense	1000	400	0.08	0.03	0.40	90.15
I dense	1000	500	0.03	0.00	0.20	131.65
II sparse	3000	900	1.09	0.05	1.52	161.18
II sparse	3000	1200	0.59	0.03	1.02	238.37
II sparse	3000	1500	0.30	0.00	0.57	197.37
III sparse	1000	300	1.95	0.07	5.19	140.47
III sparse	1000	400	1.25	0.04	2.36	104.16
III sparse	1000	500	0.77	0.02	1.40	157.43

Email graph represents email network of $n = 1133$ members of the Univeristy Rovira i Virgili (Tarragona), see [27]. There are $m = 5451$ edges in the graph. From [51] we know that email graph has a clique with 12 vertices. Our algorithm q-RCC2 with $q = 40$ finds the clique in 6 s (!).

Our final set of experiments consider graphs from the collaboration networks database. CA-GrQc collaboration network from [38] covers scientific collaborations between authors of papers that are submitted to General Relativity and Quantum Cosmology category. The data covers papers in the period of 124 months i.e. from January 1993 to April 2003. The adjacency matrix of the graph has 5242 vertices and 14,496 edges. We are not aware of an optimal value for the densest k-subgraph problem on CA-GrQc. Therefore, we present our results for different k, see Table 6. In the row denoted by CA-GrQc we list for each k the best computed objective value and the corresponding computational time in seconds. All results are obtained using the same settings: 200 simultaneous updates and 3000 iterations per round. Note that for $k = 10,20,30,40$ we find the optimal cliques. In all those cases we needed to restart the algorithm at most 4 times. For $k = 50$ we could not find a clique, and the best solution found is equal to 2146.

Further we provide similar experiments for CA-HepTh collaboration network, see [38]. This network covers scientific collaborations between authors of papers submitted to High Energy Physics – Theory category. The adjacency matrix of this graph is of order 9877. There are 25,998 edges in this network. We set $q = 1000$ and look for the densest k-subgraph in the graph. Again, we are not aware of an optimal value for the densest k-subgraph problem on CA-GrQc. Our computational results are given in Table 6. We find cliques for $k = 10,20,30$.

CA-HepPh collaboration network considers scientific collaborations between authors of papers that are submitted to High Energy Physics – Phenomenology category, see [38]. The data covers papers in the period of 124 months, i.e. from January 1993 to April 2003. The resulted graph has 12,008 vertices and 118,521 edges. The results for the densest k-subgraph problem on CA-HepPh for $k = 10,20,40,50$ are given in Table 6. It is remarkable that we can found cliques with 10, 20, 30, 40 and 50 vertices in short time. We use here $q = 1000$.

CA-AstroPh collaboration network covers scientific collaborations between authors of papers submitted to Astro Physics category, see [38]. The data covers papers in the period from January 1993 to April 2003. The adjacency matrix of the graph has 18,772 rows and 118,521 edges. We take here $q = 1000$. Our computational results show that for larger q the computations are too expensive. We find cliques for each k in less than 13 min.

Finally, we consider CA-CondMat collaboration network. This collaboration network considers scientific collaborations between 23,133 authors whose papers are submitted to Condense Matter category. The resulted adjacency matrix has 93,497 edges. The data cover papers in the period from January 1993 to April 2003. In CA-CondMat we found a clique

Table 6. Results obtained by q-RCC2 for different k.

k	10	20	30	40	50
CA-GrQc	90 (0.3)	380 (0.4)	870 (0.6)	1560 (5.9)	2146 (9.1)
CA-HepTh	90 (35.4)	380 (48.2)	870 (152.5)	1048 (176.7)	1166 (396.2)
CA-HepPh	90 (212.0)	380 (370.9)	870 (410.1)	1560 (74.8)	2450 (38.6)
CA-AstroPh	90 (369.5)	380 (180.6)	870 (529.5)	1560 (447.5)	2450 (748.4)

with 10 vertices in 159 s, and a clique with 25 vertices in 211 s. To find densest subgraphs we set $q = 1300$.

6. Conclusion

There are many studies on random coordinate descent algorithms for convex problems, but a very few results on solving non-convex large scale problems. In this paper we present two algorithms for solving large scale non-convex problems with one linear constraint. We exploit our algorithms to solve large scale instances of the densest k-subgraph problem.

The main difference between our algorithms and those in the literature is that we allow updating more than two random coordinates simultaneously in each iteration of the algorithms. Our numerical results demonstrate significant improvement in bounds for larger than two simultaneous updates of the algorithms, see Tables 1–4. The q-RCC2 algorithm performs better than q-RCC1, and for an appropriate q it converges to an integer solution of the problem. Note that the q-RCC2 algorithm considers a linear approximation of the non-convex objective function.

Our numerical results verify the efficiency of the here introduced approach. For instance, we are able to find densest k-subgraphs in real world graphs with up to 23,133 vertices in a few minutes. Our numerical results in Section 5 can be used as a benchmark for solving the densest k-subgraph problem on large graphs.

Acknowledgements

The author would like to thank Pavel Dvurechensky for useful discussions on the random coordinate descent approaches. The author would also like to thank two anonymous referees for suggestions that led to an improvement of this paper.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

[1] B.P. Ames, *Guaranteed recovery of planted cliques and dense subgraphs by convex relaxation*, J. Optim. Theory Appl. 167(2) (2015), pp. 653–675.
[2] B.P. Ames and S.A. Vavasis, *Nuclear norm minimization for the planted clique and biclique problems*, Math. Program. 129 (2011), pp. 69–89.
[3] K.M. Anstreicher, *Linear programming in $O([n^3/\ln n]L)$ operations*, SIAM J. Optim. 9(4) (1999), pp. 803–812.
[4] R. Andersen and K. Chellapilla, *Finding dense subgraphs with size bounds*, in *WAW'09: Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, K. Avrachenkov, D. Donato, and N. Litvak, eds., Barcelona, Spain, February 12–13, 2009, 25–37, Springer, Heidelberg, Berlin, 2009.
[5] A. Angel, N. Sarkas, K. Koudas, and D. Srivastava, *Dense subgraph maintenance under streaming edge weight updates for real-time story identification*, Proc. VLDB Endowment 5(6) (2012), pp. 574–585.
[6] S. Arora, D. Karger, and M. Karpinski, *Polynomial time approximation schemes for dense instances of NP-hard problems*, J. Comput. Syst. Sci. 58(1) (1999), pp. 193–210.

[7] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, *Detecting high log-densities: An $\mathcal{O}(n^{1/4})$ approximation for densest k-subgraph*, in *STOC '10: Proceedings of the Forty-second ACM Symposium on Theory of Computing*, M. Mitzenmacher and L.J. Schulman, eds., Cambridge, MA, USA, June 6–8, 2010, ACM, New York, NY, USA, 2010, pp. 201–210).

[8] A. Billionnet, *Different formulations for solving the heaviest k-subgraph problem*, INFOR: Inf. Syst. Oper. Res. 43(3) (2005), pp. 171–186.

[9] A. Billionnet, S. Elloumi, and M.C. Plateau, *Improving the performance of standard solvers for quadratic 0–1 programs by a tight convex reformulation: The QCR method*, Discrete Appl. Math. 157(6) (2009), pp. 1185–1197.

[10] A. Billionnet, S. Elloumi, A. Lambert, and A. Wiegele, *Using a conic bundle method to accelerate both phases of a quadratic convex reformulation*, INFORMS J. Comput. 29(2) (2017), pp. 318–331.

[11] M. Bougeret, N. Bousquet, R. Giroudeau, and R. Watrigant, *Parameterized complexity of the sparsest k-subgraph problem in chordal graphs*, in *SOFSEM 2014: Theory and Practice of Computer Science: 40th International Conference on Current Trends in Theory and Practice of Computer Science*, V. Geffert, B. Preneel, B. Rovan, Jú. Štuller, and A.M. Tjoa, eds., Smokovec, Slovakia, January 26–29, 2014, Springer, Switzerland, 2014, pp. 150–161.

[12] N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, and V.Th. Paschos, *Exact and superpolynomial approximation algorithms for the densest k-subgraph problem*, EJOR 262 (2017), pp. 894–903.

[13] J. Brimberg, N. Mladenović, D. Urošević, and E. Ngai, *Variable neighborhood search for the heaviest k-subgraph*, Comput. Oper. Res. 36(11) (2009), pp. 2885–2891.

[14] M. Bruglieri, M. Ehrgott, H.W. Hamacher, and F. Maffioli, *An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints*, Discrete Appl. Math. 154(9) (2006), pp. 1344–1357.

[15] D.G. Corneil and Y. Perl, *Clustering and domination in perfect graphs*, Discrete Appl. Math. 9(1) (1984), pp. 27–39.

[16] Y. Dourisboure, F. Geraci, and M. Pellegrini, *Extraction and classification of dense communities in the web*, in *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, C. Williamson, M.E. Zurko, P. Patel-Schneider, and P. Shenoy, eds., Banff, Alberta, Canada, May 8–12, 2007, ACM, New York, NY, USA, 2007, pp. 461–470.

[17] P. Erdös and A. Rényi, *On random graphs*, Publ. Math. 6 (1959), pp. 290–297.

[18] P. Erdös and R. Rényi, *On the evolution of random graphs*, Publ. Math. Inst. Hungar. Acad. Sci. 5 (1960), pp. 17–61.

[19] U. Feige, G. Kortsarz, and D. Peleg, *The dense k-subgraph problem*, Algorithmica 29(3) (2001), pp. 410–421.

[20] U. Feige and M. Langberg, *Approximation algorithms for maximization problems arising in graph partitioning*, J. Algorithms 41 (2001), pp. 174–211.

[21] U. Feige and M. Seltser, *On the densest k-subgraph problem*, Technical Report CS97-16, Weizmann Institute of Science, Rehovot, Israel, 1997.

[22] A. Gajewar and A. Das arma, *Multi-skill collaborative teams based on densest subgraphs*, in *Proceedings of the 2012 SIAM International Conference on Data Mining*, J. Ghosh, H. Liu, I. Davidson, C. Domeniconi, and C. Kamath, eds., Anaheim, CA, USA, April 26–28, SIAM, Philadelphia, PA, USA, 2012, pp. 165–176.

[23] D. Gibson, R. Kumar, and A. Tomkins, *Discovering large dense subgraphs in massive graphs*, in *VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases*, K. Bohm, C.S. Jensen, L.M. Haas, M.L. Kersten, P.A. Larson, and B.C. Ooi, eds., August 30–September 2, ACM, Trondheim, Norway, 2005, pp. 721–732.

[24] P. Gleiser and L. Danon, *Community structure in jazz*, Adv. Complex Syst. 6 (2003), pp. 565.573

[25] O. Goldschmidt and D.S. Hochbaum, *k-edge subgraph problems*, Discrete Appl. Math. 74(2) (1997), pp. 159–169.

[26] D. Goldfarb and S. Liu, *An $O(n^3L)$ primal interior point algorithm for convex quadratic programming*, Math. Program. 49 (1991), pp. 325–340.

[27] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas, *Self-similar community structure in a network of human interactions*, Phys. Rev. E 68 (2003), pp. 065103(R).
[28] M.R. Henzinger, R. Motwani, and C. Silverstein, *Challenges in web search engines*, IJCAI'03: Proceedings of the 18th International Joint Conference on Artificial Intelligence, August 9–15, Vol. 3, Morgan Kaufmann Publishers Inc., Acapulco, Mexico, 2003, pp. 1573–1579.
[29] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica 4 (1984), pp. 373–395.
[30] J.M. Keil and T.B. Brecht, *The complexity of clustering in planar graphs*, J. Combin. Math. Combin. Comput. 9 (1991), pp. 155–159.
[31] L.G. Khachiyan, *A polynomial algorithm in linear programming*, Soviet Math. Dokl. 20 (1979), pp. 191–194. (Dokl. Akad. Nauk SSSR244), 1093–1096.
[32] S. Khot, *Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique*, SIAM J. Comput. 36(4) (2006), pp. 1025–1071.
[33] R.K. Kincaid, *Good solutions to discrete noxious location problems via metaheuristics*, Ann. Oper. Res. 40(1) (1992), pp. 265–281.
[34] M. Kojima, S. Mizuno, and A. Yoshise, *A polynomial-time algorithm for a class of linear complementarity problems*, Math. Program. 44 (1988), pp. 1–26.
[35] G. Kortsarz and D. Peleg, *On choosing a dense subgraph*, Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, November 3–5, Palo Alto, CA, USA, 1993, pp. 692–701.
[36] M.K. Kozlov, S.P. Tarasov, and L.G. Khachiyan, *Polynomial solvability of convex quadratic programming*, Doklady Akademiia Nauk SSSR 248 (1979) (translated in Soviet Mathematics Doklady, 20, 1108–111).
[37] N. Krislock, J. Malick, and F. Roupin, *Computational results of a semidefinite branch-and-bound algorithm for k-cluster*, Comput. Oper. Res. 66 (2016), pp. 153–159.
[38] J. Leskovec, J. Kleinberg, and C. Faloutsos, *Graph evolution: Densification and shrinking diameters*, ACM Trans. Knowledge Discovery Data (ACM TKDD) 1 (2007), pp. 1–41.
[39] X. Li, Y. Chen, and J. Xu, *Convex relaxation methods for community detection*, 2018. arXiv preprint arXiv:1810.00315.
[40] J. Malick and F. Roupin, *Solving k-cluster problems to optimality with semidefinite programming*, Math. Program. Ser. B 136 (2012), pp. 279–300.
[41] E.M. Macambira, *An application of tabu search heuristic for the maximum edge-weighted subgraph problem*, Ann. Oper. Res. 117(1) (2002), pp. 175–190.
[42] Yu. Nesterov, *Effciency of coordinate descent methods on huge-scale optimization problems*, SIAM. J. Optim. 22(2) (2012), pp. 341–362.
[43] Yu. Nesterov, *Subgradient methods for huge-scale optimization problems*, Math. Program. 146(1–2) (2014), pp. 275–297.
[44] I. Necoara, Yu Necoara, and F. Glineur, *Random block coordinate descent methods for linearly constrained optimization over networks*, J. Optim. Theory. Appl. 173(1) (2017), pp. 227–254.
[45] T. Nonner, *PTAS for densest k-subgraph in interval graphs*, Algorithmica 74(1) (2016), pp. 528–539.
[46] A. Patrascu and I. Necoara, *Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization*, J. Global Optim. 61(1) (2015), pp. 19–46.
[47] S.S. Ravi, D.J. Rosenkrantz, and G.K. Tayi, *Heuristic and special case algorithms for dispersion problems*, Oper. Res. 42(2) (1994), pp. 299–310.
[48] F. Rendl, *Semidefinite relaxations for partitioning, assignment and ordering problems*, Ann. Oper. Res. 240 (2016), pp. 119–140.
[49] P. Richtárik and M. Takáč, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Math. Program. 144(1-2) (2014), pp. 1–38.
[50] C. Tsourakakis, *The k-clique densest subgraph problem*, in *Proceedings of the 24th International Conference on World Wide Web*, A. Gangemi, S. Leonardi, and A. Panconesi, eds., May 18–22, ACM, Florence, Italy, 2015, pp. 1122–1132.
[51] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, *Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees*, Proceedings of the 19th ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 104–112.
[52] S. Van der Doef, *Applying huge-scale coordinate descent methods to the densest k-subgraph problem*, Master thesis, Tilburg University, Tilburg, 2017.
[53] Y. Ye and J. Zhang, *Approximation of dense-n/2 subgraph and the complement of min-bisection*, J. Global Optim. 25 (2003), pp. 55–73.