Optimization Letters (2019) 13:1211–1222
https://doi.org/10.1007/s11590-019-01418-9

ORIGINAL PAPER

Algorithms for the generalized independent set problem based on a quadratic optimization approach

Seyedmohammadhossein Hosseinian[1] · Sergiy Butenko[1]

Received: 14 September 2018 / Accepted: 7 March 2019 / Published online: 15 March 2019

Abstract
This paper presents two algorithms—a construction heuristic and an exact solution method—for the generalized independent set problem. Both methods take advantage of a nonlinear formulation of this problem and are based on optimization of a quadratic function over a hypersphere. Heuristic solutions are constructed by exploring stationary points of a surrogate problem of this form. The exact method is a combinatorial branch-and-bound algorithm that uses a spherical relaxation of the original feasible region in its pruning subroutine. We show competence of the proposed methods through computational experiments on benchmark instances.

Keywords Generalized independent set problem · Quadratic optimization · Combinatorial branch-and-bound · Construction heuristic

1 Introduction

Consider a simple and undirected graph $G = (V, E)$, where V is the vertex set and $E = E_1 \cup E_2$ is the edge set partitioned into the set of *permanent* edges E_1 and the set of *removable* edges E_2. Each vertex $i \in V$ has a benefit $b_i > 0$ and each removable edge $\{i, j\} \in E_2$ may be eliminated at a cost $c_{ij} > 0$. Note that, by definition of permanent edges, c_{ij} is not defined for an edge $\{i, j\} \in E_1$. The generalized independent set problem (GISP) is to find an independent set $I \subseteq V$ (i.e., a subset of vertices with no edge connecting any two vertices among them) with the maximum net benefit, that is the total benefit of the vertices in I minus the total cost of deleting removable edges connecting them. Clearly, the classical maximum (cardinality) independent set problem is a special case of GISP with $E_2 = \emptyset$ and $b_i = 1,\ \forall i \in V$; hence the name.

✉ Seyedmohammadhossein Hosseinian
hosseinian@tamu.edu

Sergiy Butenko
butenko@tamu.edu

1 Texas A&M University, College Station, TX, USA

This problem was first introduced by Hochbaum and Pathria [10] in the context of forest harvesting. They proposed an integer (linear) programming formulation for it and showed that GISP is polynomial-time solvable on bipartite graphs. In [9], Hochbaum revisited this problem and presented it as an extension of the maximum closure problem with application in facility location. She also further studied the integer (linear) programming formulation of GISP in [8], and proposed a procedure that generates a half-integral superoptimal solution with the objective value tighter than the corresponding LP relaxation bound. Later, Kochenberger et al. [16] used an unconstrained binary quadratic formulation of GISP to develop a tabu search algorithm for it. Very recently, Colombi et al. [6] conducted a polyhedral study and proposed an exact branch-and-cut algorithm for this problem as well as three heuristic methods. Results of their extensive computational experiments revealed that GISP instances are generally very challenging for the state-of-the-art MIP solvers, even with the help of the newly identified strong valid inequalities. Motivated by this observation, Kalil et al. [15] used this problem as a testbed to evaluate performance of the MIP solvers equipped with a supervised-learning oracle. Their results also confirm computational difficulty of solving GISP as well as finding high-quality solutions. To the best of our knowledge, these are the only works addressing applications and solution methods of the generalized independent set problem in the literature.

We note that GISP casts a variant of the two-stage stochastic maximum (vertex) weight independent set problem (MWIS), which finds applications in image/video processing [3,11], wireless network operation [1,18] and computational biology [4] among others. Consider a graph $G = (V, E)$ with some uncertainty about authenticity of the edges. This may be the case due to using partial or noisy data in construction of the graph in practice. Suppose that an edge $\{i, j\} \in E$ actually exists with a probability p_{ij}, or equivalently, it is a false inclusion with a probability $1 - p_{ij}$. Given a positive weight (benefit) $b_i > 0$ associated with each vertex $i \in V$, MWIS is to find an independent set with the maximum total weight in G. In the two-stage stochastic optimization setting, it is assumed that the uncertainty is resolved after the decision is made (i.e., a subset of vertices is selected), then a *recourse* action is allowed to satisfy the violated constraints at predefined costs. Under this setting, the two-stage stochastic version of MWIS is to find a subset of vertices with the maximum difference between the total benefit and the *expected* total recourse cost. Given a predefined cost $C_{ij} > 0$ to remedy the dependence of two selected vertices i and j, the expected recourse cost associated with this violation will be $c_{ij} = p_{ij}C_{ij}$. The set of edges for which a recourse action is not allowed or those with $c_{ij} > \min\{b_i, b_j\}$ are considered as the set of permanent edges. Then, it becomes clear that GISP formulates the described form of the two-stage stochastic MWIS. We refer to [2] for more details on the two-stage stochastic programming.

In this paper, we propose two solution methods—a construction heuristic and an exact algorithm—for GISP. These methods are extensions of the algorithms proposed for the maximum edge weight clique problem in [12,13] and take advantage of a quadratic formulation of GISP. We solve a surrogate of the original formulation to find high-quality feasible solutions for GISP. Besides, we calculate an upper bound on its optimal objective value by solving a quadratic relaxation of this formulation. This bound is then employed in a combinatorial branch-and-bound (B&B) procedure and an exact algorithm for GISP is developed. The remainder of this paper is organized as follows: Sect. 2 presents the heuristic method. Section 3 describes the combinatorial B&B algorithm, including calculation of the upper bound and the pruning subroutine. Section 4 provides results of computational experiments, and Sect. 5 concludes this paper.

2 Construction heuristic method

Consider an instance of GISP defined on a simple and undirected graph $G = (V, E_1 \cup E_2)$, where $V = \{1, \dots, n\}$. Let $\mathbf{b} = (b_1, \dots, b_n)^\top$ be the vector of benefits associated with the vertices of G, and $\mathbf{Q}$ be an $n \times n$ matrix with the following structure:

$$\mathbf{Q}(i,j) = \begin{cases} \min\{b_i, b_j\} + \varepsilon, & \{i,j\} \in E_1 \\ c_{ij}, & \{i,j\} \in E_2 \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where ε denotes an arbitrarily small positive number. The following proposition establishes the quadratic formulation of GISP that is used in this paper to develop heuristic and exact solution methods for this problem.

Proposition 1 *The generalized independent set problem on G can be formulated as the following unconstrained binary quadratic program:*

$$\max_{\mathbf{x} \in \{0,1\}^n} f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} - \frac{1}{2} \mathbf{x}^\top \mathbf{Q} \mathbf{x}. \tag{2}$$

Proof Let $I^* \subseteq V$ be an optimal solution to GISP on G, and $\mathbf{x}^*$ denote an optimal solution of (2). We first show that $f(\mathbf{x}^*) \leq b(I^*)$, where $b(I^*)$ denotes the net benefit of I^*. To this end, let $\tilde{I} = \{i \in V \mid x_i^* = 1\}$. We claim that there is no permanent edge in $G[\tilde{I}]$, the subgraph induced by $\tilde{I}$. Suppose that this is not the case and there is a permanent edge $\{i, j\} \in E_1$ connecting two vertices $i, j \in \tilde{I}$. Without loss of generality, let $\min\{b_i, b_j\} = b_j$. Then, changing the value of x_j^* from 1 to 0 will increase the objective function value of (2), which contradicts optimality of $\mathbf{x}^*$. Thus, $\tilde{I}$ is an independent set in $G_1 = (V, E_1)$, and a feasible solution to GISP. Besides, given the construction of $f(\mathbf{x})$, the net benefit of $\tilde{I}$ equals $f(\mathbf{x}^*)$, which immediately leads to $f(\mathbf{x}^*) = b(\tilde{I}) \leq b(I^*)$. To complete the proof, we need to show that $b(I^*) \leq f(\mathbf{x}^*)$ also holds. Let $\tilde{\mathbf{x}}$ denote the incidence vector of I^*, that is $\tilde{x}_i = 1, \forall i \in I^*$, and $\tilde{x}_i = 0$ otherwise. Patently, $b(I^*) = f(\tilde{\mathbf{x}})$ by definition of the function f, which—along with feasibility of $\tilde{\mathbf{x}}$ to (2)—implies $b(I^*) = f(\tilde{\mathbf{x}}) \leq f(\mathbf{x}^*)$. Therefore, $f(\mathbf{x}^*) = b(I^*)$ and $\mathbf{x}^*$ characterizes an independent set with the maximum net benefit in G. □

Given the binary assignment of the variables (i.e., as $x_i = x_i^2, \ \forall i \in V$), we may rewrite (2) as optimization of a quadratic form, as follows:

$$\max_{\mathbf{x} \in \{0,1\}^n} \frac{1}{2} \mathbf{x}^\top (2\mathbf{B} - \mathbf{Q}) \mathbf{x}, \tag{3}$$

where $\mathbf{B} = \operatorname{diag}(b_1, \ldots, b_n)$ is a square matrix of order n with $\mathbf{b}$ on its main diagonal and zero as the off-diagonal entries. Suppose that an optimal solution of GISP on G consists of k vertices, hence the corresponding optimal solution of (3) is located on the hypersphere $\mathbf{x}^\top \mathbf{x} = k$. By replacing the discrete feasible region of (3) with this hypersphere and a change of variables $\mathbf{y} = \frac{1}{\sqrt{k}}\mathbf{x}$, we get the following problem as a surrogate of (3):

$$\begin{aligned} \max_{\mathbf{y} \in \mathbb{R}^n} \quad & \frac{k}{2} \mathbf{y}^\top (2\mathbf{B} - \mathbf{Q})\, \mathbf{y} \\ \text{s.t.} \quad & \mathbf{y}^\top \mathbf{y} = 1. \end{aligned} \tag{4}$$

We construct a heuristic solution for GISP via exploring stationary points of (4). It is well known that the stationary points of (4) are given by (normalized) eigenvectors of matrix $\mathbf{Q}' = 2\mathbf{B} - \mathbf{Q}$ [17]. We take each eigenvector of this matrix as an estimate of a binary solution to detect two independent sets in the graph. Detecting an independent set corresponding to an eigenvector $\mathbf{x}_e$ of $\mathbf{Q}'$ is done as follows: first, the vertices are sorted in a *descending* order of their incidence values in $\mathbf{x}_e$. An independent set is constructed by appending the vertices one-by-one from the ordered list. A new vertex is added to an incumbent independent set I if it is not adjacent to any vertex of I in $G_1 = (V, E_1)$ and its benefit is strictly greater than the sum of the costs of the removable edges connecting it to the vertices in I. Clearly, in this phase of the algorithm, higher priorities are given to the vertices whose incidence values in $\mathbf{x}_e$ are closer to 1. As $\mathbf{x}_e$ entries are signed, the algorithm also finds an independent set using the reverse of this order by sorting the vertices in an *ascending* order of the incidence values in $\mathbf{x}_e$. It should be mentioned that these two independent sets are not necessarily distinct.

Clearly, every optimal solution of GISP on G must be an independent set in $G_1 = (V, E_1)$. Therefore, given that an optimal solution contains a certain vertex $i \in V$, one may restrict the search to $G^i = G[V \backslash N_1(i)]$, the subgraph induced by $V \backslash N_1(i)$, where $N_1(i)$ denotes the set of vertices adjacent to i in G_1. Based on this observation, our heuristic algorithm performs the described process n times, each time assuming a fixed vertex i belongs to the optimal solution. The algorithm returns an independent set with the maximum net benefit among all the detected ones. Algorithm 1 outlines the procedure. We have used $b(I)$ to denote the net benefit of an independent set I in this algorithm.

3 Combinatorial branch-and-bound method

We take advantage of the fact that every optimal solution of GISP is an independent set in $G_1 = (V, E_1)$ to enumerate them implicitly, and use a *quadratic relaxation* of (2) to prune the search space. The systematic search is done by a recursive function, called Search, which inputs an ordered list of vertices. In this list, vertices $i \in V$ are sorted in a descending order of their degrees in G_1, intending to avoid large subproblems. We employ the enumeration technique, including the initial ordering of the vertices, that was originally proposed by Carraghan and Pardalos [5]. Algorithm 2 outlines the exact method.

Algorithm 1 Construction heuristic for GISP

```
1: function GISP_H($G$)
2:     $I^* = \emptyset$ ;  $b^* = 0$
3:     for each vertex $i \in V$ do
4:         $G^i = G[V \backslash N_1(i)]$
5:         construct the matrix $\mathbf{Q}' = 2\mathbf{B} - \mathbf{Q}$ corresponding to $G^i$
6:         for each eigenvector $\mathbf{x}_e$ of $\mathbf{Q}'$ do
7:             $I_d$ = DETECTIS($\mathbf{x}_e$, $descending$, $i$, $G^i$)
8:             if $b(I_d) > b^*$ then                                   ▷ $b(I_d)$: net benefit of $I_d$
9:                 $I^* \leftarrow I_d$ ;  $b^* \leftarrow b(I_d)$
10:            end if
11:            $I_a$ = DETECTIS($\mathbf{x}_e$, $ascending$, $i$, $G^i$)
12:            if $b(I_a) > b^*$ then                                   ▷ $b(I_a)$: net benefit of $I_a$
13:                $I^* \leftarrow I_a$ ;  $b^* \leftarrow b(I_a)$
14:            end if
15:        end for
16:    end for
17:    return $(I^*, b^*)$
18: end function
19: –
20: function DETECTIS($\mathbf{x}$, $order$, $v$, $G$)
21:    sort entries of $\mathbf{x}$ according to $order$                 ▷ sorted array: $[x^{(1)}, \ldots, x^{(n)}]$
22:    $I = \{v\}$
23:    for $j = 1$ to $n$ do
24:        $u = ver[x^{(j)}]$                                           ▷ $ver[x^{(j)}]$: vertex corresponding to $x^{(j)}$
25:        if $u \notin \bigcup_{i \in I} N_1(i)$ and $b_u > \sum_{i \in I} c_{iu}$ then
26:            $I \leftarrow I \cup \{u\}$
27:        end if
28:    end for
29:    return $I$
30: end function
```

Algorithm 2 Exact method for GISP

```
1: function GISP_E($G$)
2:     $I = I^* = \emptyset$ ;  $b = b^* = 0$
3:     $L$ = sorted list of vertices in a descending order of their degrees in $G_1$
4:     SEARCH($G, L, I, b, I^*, b^*$)
5:     return $(I^*, b^*)$
6: end function
```

The SEARCH function operates as follows: at each call, this function inputs an independent set I and a list of *candidate* vertices $L = V \backslash \bigcup_{i \in I} N_1(i)$ associated with I. At the beginning of the search, $I = \emptyset$ and L contains all vertices of the graph. First, the PRUNE function is called to determine whether $G[I \cup L]$, the subgraph induced by $I \cup L$, should be processed further. If so, I is expanded by adding the first vertex in L, denoted by v in Algorithm 3. Then, the candidate list associated with $I \cup \{v\}$ is built, and the function works on the corresponding subproblem recursively. Patently, the function returns after all independent sets containing $I \cup \{v\}$ are examined, thus the vertex v is removed from both the expanded independent set and its candidate list upon solving this subproblem. Algorithm 3 illustrates the search process.

Pruning the search tree is done via calculating an upper bound on the maximum net benefit of an independent set contained in $G[I \cup L]$. If such an upper bound is

Algorithm 3 Combinatorial branch-and-bound procedure

```
1: function SEARCH($G, L, I, b, I^*, b^*$)
2:    while $L \neq \emptyset$ do
3:       $r$ = PRUNE($G, L, I, b, b^*$)
4:       if $r$ = true then
5:          return
6:       else
7:          $v \leftarrow L[1]$
8:          $\delta = b_v - \sum_{i \in I} c_{iv}$
9:          $I \leftarrow I \cup \{v\}$ ;  $b \leftarrow b + \delta$
10:          if $b > b^*$ then
11:             $I^* \leftarrow I$ ;  $b^* \leftarrow b$
12:          end if
13:          $L_v = L \backslash N_1(v)$                 ▷ $L_v$ keeps the relative order of vertices in $L$
14:          SEARCH($G, L_v, I, b, I^*, b^*$)
15:          $I \leftarrow I \backslash \{v\}$ ;  $b \leftarrow b - \delta$
16:       end if
17:       $L \leftarrow L \backslash \{v\}$
18:    end while
19:    return
20: end function
```

not greater than the net benefit of the incumbent solution (b^*), further search on this subgraph will never lead to an objective value larger than b^*, so it should be excluded from our search. Consider the quadratic formulation (2) of GISP on this subgraph. Note that construction of $G[I \cup L]$ is based on the assumption that I is a part of the optimal solution, hence $x_i = 1$ for every $i \in I$. Let $q_\ell = b_\ell - \sum_{i \in I} c_{i\ell}, \ \forall \ell \in L$. Then, the optimal objective value of GISP on $G[I \cup L]$ is obtained by:

$$b(I) + \max_{\mathbf{x} \in \{0,1\}^{|L|}} \mathbf{q}^\top \mathbf{x} - \frac{1}{2} \mathbf{x}^\top \mathbf{Q}_L \, \mathbf{x}, \tag{5}$$

where $b(I)$ denotes the net benefit of I, $\mathbf{q} = (q_1, \ldots, q_{|L|})^\top$, and $\mathbf{Q}_L$ is the principal submatrix of $\mathbf{Q}$ corresponding to the vertices of L. We use the following relaxation of (5), by replacing its discrete feasible region with a single hypersphere, to get an upper bound on its optimal value:

$$b(I) + \left(\begin{array}{ll} Z_L = \max\limits_{\mathbf{x} \in \mathbb{R}^{|L|}} & \mathbf{q}^\top \mathbf{x} - \frac{1}{2} \mathbf{x}^\top \mathbf{Q}_L \, \mathbf{x} \\ \quad \text{s.t.} & (\mathbf{x} - \frac{1}{2}\mathbf{1})^\top (\mathbf{x} - \frac{1}{2}\mathbf{1}) = \frac{|L|}{4} \end{array} \right), \tag{6}$$

where $\mathbf{1}$ denotes the vector of all ones.

It is a well-known result that optimization of a quadratic function subject to a single hypersphere constraint, under some regulatory conditions, reduces to finding the (unique) root of a univariate function in a known interval [7], which can be calculated fast by line search methods. Let λ_i be the i-th eigenvalue of $\mathbf{Q}_L$, $\mathbf{E}$ be a square matrix whose i-th column is the eigenvector corresponding to λ_i, and λ^{max} denote the largest eigenvalue of $\mathbf{Q}_L$. If there exists $\lambda_i = \lambda^{max}$ such that $\lambda_i \sigma_i + q'_i \neq 0$, then,

$$Z_L = \sum_{i=1}^{|L|} \frac{(\mu^* \sigma_i + q_i') \left[\lambda_i (\mu^* \sigma_i + q_i') + 2\mu^* q_i'\right]}{2(\lambda_i - \mu^*)^2}, \tag{7}$$

where q_i' and σ_i denote the i-th components of $\mathbf{q}' = \mathbf{E}^\top \mathbf{q}$ and $\boldsymbol{\sigma} = \frac{1}{2}\mathbf{E}^\top \mathbf{1}$, respectively, and μ^* is the unique root of the following function in the interval $(\lambda^{max}, +\infty)$:

$$\varphi(\mu) = \sum_{i \in \mathscr{I}} \frac{(\lambda_i \sigma_i + q_i')^2}{(\lambda_i - \mu)^2} - \frac{|L|}{4}, \tag{8}$$

where $\mathscr{I} = \{i \mid \lambda_i \sigma_i + q_i' \neq 0\}$. We use (7) in the pruning subroutine of our algorithm, as illustrated by Algorithm 4. Clearly, $b(I) + \sum_{\ell \in L} b_\ell$ is a trivial upper bound on the optimal value of (5). In case that $\lambda_i \sigma_i + q_i' = 0,\ \forall \lambda_i = \lambda^{max}$, or if the calculated value for Z_L exceeds $\sum_{\ell \in L} b_\ell$, this trivial bound is used in the pruning process of the algorithm. The frequency of such events was negligible in our computational experiments.

Algorithm 4 Pruning method

```
1: function PRUNE($G, L, I, b, b^*$)
2:     $\bar{b} = \sum_{\ell \in L} b_\ell$
3:     for each vertex $\ell \in L$ do
4:         $q_\ell = b_\ell - \sum_{i \in I} c_{i\ell}$
5:     end for
6:     for $i = 1$ to $|L|$ do
7:         $\lambda_i$ = the $i$-th smallest eigenvalue of $\mathbf{Q}_L$
8:         $\mathbf{e}^i$ = the $i$-th eigenvector of $\mathbf{Q}_L$ corresponding to $\lambda_i$
9:         $q_i' = \sum_{j=1}^{|L|} e_j^i\, q_j$
10:        $\sigma_i = \frac{1}{2} \sum_{j=1}^{|L|} e_j^i$
11:    end for
12:    $\lambda^{max} \leftarrow \lambda_{|L|}$
13:    $i \leftarrow |L|$
14:    do
15:        if $\lambda_i \sigma_i + q_i' \neq 0$ then
16:            $\mu^* = \{\mu \in (\lambda^{max}, +\infty) \mid \varphi(\mu) = 0\}$
17:            $Z_L = \sum_{i=1}^{|L|} \frac{(\mu^* \sigma_i + q_i') \left[\lambda_i (\mu^* \sigma_i + q_i') + 2\mu^* q_i'\right]}{2(\lambda_i - \mu^*)^2}$
18:            $\bar{b} \leftarrow \min\{Z_L, \bar{b}\}$
19:            if $\bar{b} \leq b^* - b$ then
20:                return true
21:            else
22:                return false
23:            end if
24:        end if
25:        $i \leftarrow i - 1$
26:    while $\lambda_i = \lambda^{max}$
27:    if $\bar{b} \leq b^* - b$ then
28:        return true
29:    else
30:        return false
31:    end if
32: end function
```

4 Computational experiments

This section presents the results of our computational experiments with the benchmark instances introduced in [6]. Colombi et al. [6] used 12 graphs from the DIMACS set [14] to generate $216 = 12\times3\times6$ instances for GISP with three different partitions of permanent/removable edges and six different parameter values (i.e., benefit and cost). We take $36 = 12 \times 3 \times 1$ instances from this set. In generation of these instances, every edge of the original graph has been randomly marked as "removable" with a probability p (or "permanent" otherwise) independent of the others. The instances correspond to $p = 0.25$, 0.50 and 0.75. Regarding the parameter values, we use those referred to as "SET1- C" in [6]. These are the instances with the most diverse parameter values. In these graphs, every vertex has been randomly assigned a benefit value in $\{1, \dots, 100\}$ and the cost associated with each removable edge $\{i, j\} \in E_2$ is given by $c_{ij} = \frac{1}{25}(b_i + b_j)$. Characteristics of the test instances are presented in Table 1. In this table, ρ_1 and ρ_2 denote the permanent and removable edge densities of each instance for the corresponding value of p, respectively. These graphs are publicly available at http://or-dii.unibs.it/index.php?page=instances.

First, we report our results for the combinatorial B&B algorithm and compare its performance with the exact solution method proposed in [6]. Colombi et al. [6] considered an integer (linear) programming formulation of GISP and identified three types of strong valid inequalities for its polytope. They used these inequalities to strengthen the integer programming formulation and proposed a branch-and-cut method to solve this problem. They studied the performance of the proposed method via extensive computational experiments under various implementation settings. We compare results of our algorithm with the most effective implementation of their method, referred to as "Final B&C" in [6] without an initial lower bound. The original "Final B&C" algorithm enjoys an initial lower bound obtained from a separate mataheuristic method.

Table 1 Instances characteristics

Name	$\|V\|$	|E|	$p = 0.25$		$p = 0.50$		$p = 0.75$	
			ρ_1	ρ_2	ρ_1	ρ_2	ρ_1	ρ_2
C125.9	125	6963	0.68	0.22	0.45	0.45	0.22	0.67
C250.9	250	27,984	0.67	0.23	0.45	0.45	0.23	0.67
MANN_a27	378	70,551	0.74	0.25	0.50	0.49	0.25	0.74
brock200_2	200	9876	0.37	0.12	0.25	0.25	0.12	0.37
brock400_2	400	59,786	0.56	0.19	0.37	0.38	0.18	0.56
gen200_p0.9_55	200	17,910	0.68	0.22	0.45	0.45	0.22	0.68
gen400_p0.9_75	400	71,820	0.67	0.23	0.45	0.45	0.23	0.67
hamming8-4	256	20,864	0.48	0.16	0.32	0.32	0.16	0.48
keller4	171	9435	0.49	0.16	0.33	0.32	0.17	0.48
p_hat300-1	300	10,933	0.18	0.06	0.12	0.12	0.06	0.18
p_hat300-2	300	21,928	0.37	0.12	0.25	0.24	0.12	0.36
p_hat300-3	300	33,390	0.56	0.18	0.38	0.37	0.19	0.56

Table 2 Solution time

	Name	*opt.*	CPU (s)		
			CB&B	B&C [6] with auto cut	B&C [6] w/o auto cut
$p = 0.25$	C125.9	403	0.171	14.234	5.968
	C250.9	502	2.625	784.344	1483.690
	MANN_a27	443	8.312	>	>
	brock200_2	932	56.046	155.422	382.078
	brock400_2	698	100.578	>	>
	gen200_p0.9_55	467	1.062	242.000	102.938
	gen400_p0.9_75	533	19.687	>	>
	hamming8-4	1094	46.562	267.938	199.094
	keller4	941	2.796	33.834	13.921
	p_hat300-1	2649	>	1792.340	3627.580
	p_hat300-2	2033	1150.160	588.094	606.891
	p_hat300-3	688	22.953	3190.640	3108.000
$p = 0.50$	C125.9	506	1.109	69.890	46.296
	C250.9	623	63.343	>	>
	MANN_a27	552	344.766	>	>
	brock200_2	1101	2291.560	3266.200	>
	brock400_2	–	>	>	>
	gen200_p0.9_55	597	17.609	3699.880	3247.780
	gen400_p0.9_75	651	1503.560	>	>
	hamming8-4	1184	1513.890	5825.200	5542.520
	keller4	1049	29.156	96.125	89.359
	p_hat300-1	–	>	>	>
	p_hat300-2	2263	>	7285.030	>
	p_hat300-3	851	934.906	>	>
$p = 0.75$	C125.9	644	128.813	2557.980	3255.060
	C250.9	–	>	>	>
	MANN_a27	–	>	>	>
	brock200_2	–	>	>	>
	brock400_2	–	>	>	>
	gen200_p0.9_55	–	>	>	>
	gen400_p0.9_75	–	>	>	>
	hamming8-4	–	>	>	>
	keller4	1109	9859.310	3744.280	7229.470
	p_hat300-1	–	>	>	>
	p_hat300-2	–	>	>	>
	p_hat300-3	–	>	>	>

The results are presented in Table 2. In this table, "*opt.*" denotes the optimal solution value for each instance (if known). The column "CB&B" shows the solution time of the combinatorial B&B algorithm (in CPU seconds) for each instance. The algorithm

Table 3 Heuristic results

	Name	Best found	Heuristic sol. value	CPU (s)	Gap (%)
$p = 0.25$	`C125.9`	403*	403	0.062	0.00
	`C250.9`	502*	502	0.734	0.00
	`MANN_a27`	443*	443	1.453	0.00
	`brock200_2`	932*	932	1.515	0.00
	`brock400_2`	698*	698	6.578	0.00
	`gen200_p0.9_55`	467*	467	0.343	0.00
	`gen400_p0.9_75`	533*	533	3.250	0.00
	`hamming8-4`	1094*	1094	2.265	0.00
	`keller4`	941*	941	0.609	0.00
	`p_hat300-1`	2649*	2386	13.421	9.93
	`p_hat300-2`	2033*	1740	6.578	14.41
	`p_hat300-3`	688*	688	2.500	0.00
$p = 0.50$	`C125.9`	506*	503	0.250	0.59
	`C250.9`	623*	620	2.250	0.48
	`MANN_a27`	552*	548	7.296	0.72
	`brock200_2`	1101*	1080	2.390	1.91
	`brock400_2`	892	863	16.125	3.25
	`gen200_p0.9_55`	597*	597	1.062	0.00
	`gen400_p0.9_75`	651*	645	11.437	0.92
	`hamming8-4`	1184*	1165	4.343	1.60
	`keller4`	1049*	1049	1.078	0.00
	`p_hat300-1`	2897	2562	17.796	11.56
	`p_hat300-2`	2263*	1982	10.250	12.42
	`p_hat300-3`	851*	851	5.906	0.00
$p = 0.75$	`C125.9`	644*	629	0.531	2.33
	`C250.9`	722	722†	5.421	0.00
	`MANN_a27`	626	626†	22.718	0.00
	`brock200_2`	1281	1233	3.796	3.75
	`brock400_2`	949	949†	39.109	0.00
	`gen200_p0.9_55`	727	690	2.593	5.09
	`gen400_p0.9_75`	729	729†	31.859	0.00
	`hamming8-4`	1303	1303†	7.687	0.00
	`keller4`	1109*	1060	2.031	4.42
	`p_hat300-1`	3457	3106	26.953	10.15
	`p_hat300-2`	2417	2187	17.375	9.52
	`p_hat300-3`	950	950†	12.609	0.00

was implemented in C++ and executed on an Intel® Core$^{\text{TM}}$ i7 CPU @ 2.90 GHz computer. We have set $\varepsilon = 1$ in (1), and used Intel® Math Kernel Library to perform the spectral calculations. The next two columns correspond to the solution times of the branch-and-cut method of Colombi et al. [6] using ILOG/CPLEX 12.7 solver on the same system. The column "B&C [6] with auto cut" shows the results when the automatic cut generation of the solver was on, and "B&C [6] w/o auto cut" shows the results when it was off. We restricted the solution time for each instance by each method to three hours (10,800 s) and use ">" instead of "> 10,800" in Table 2 to indicate termination of the procedure due to the time constraint.

We could solve 22 instances (out of 36) within three hours, while this number for the branch-and-cut method of Colombi et al. [6] was 17 with CPLEX automatic cuts and 15 without them. The results show that difficulty of solving GISP increases as the graph structure becomes more flexible, i.e., as density of removable edges increases. In fact, most instances corresponding to $p = 0.75$ were too hard for both methods to be solved within three hours. For instances corresponding to $p = 0.25$ and $p = 0.50$, the CB&B solution times are much better than B&C [6] in general. Among them, there are five instances solved by the CB&B algorithm in less than 6 min while the B&C [6] method failed to solve them in three hours. The CB&B algorithm performed poorly on `p_hat300-1` and `p_hat300-2`. These two are our sparsest graphs and showed the worst upper-bound quality among the test instances.

Finally, Table 3 contains the computational results for the construction heuristic algorithm. In this table, "Best found" refers to the best solution found by either of the exact methods in three hours or by the heuristic algorithm itself. The values marked by $\star$ in this column are known to be optimal. The other headings in this table are self-explanatory. We have used † to indicate that the solution found by the heuristic algorithm was strictly better than the best solution found by the exact methods within the running time. The percentage gap between the heuristic and the best-known solution values is calculated by $\frac{\text{Best}-\text{Heuristic}}{\text{Best}} \times 100$. Similar to the exact method, the best heuristic results correspond to the most rigid graphs. The algorithm attained an optimal solution on ten instances among those corresponding to $p = 0.25$. With the exception of the `p_hat` family, the percentage gap remains below 5.1% as density of the removable edges increases. As shown in Table 3, quality of the heuristic solution was strictly better than the best solution found by either of the exact methods (in three hours) for six instances corresponding to $p = 0.75$. As before, `p_hat300-1` and `p_hat300-2` are responsible for the inferior heuristic results.

5 Conclusion

The generalized independent set problem (GISP), as apparent from the name, is a generalization of the classical maximum (cardinality) independent set problem and an extension of the maximum weight independent set problem to graphs with some uncertain edges. We proposed a construction heuristic and an exact solution method for GISP based on a quadratic formulation of this problem. To find high-quality solutions, the heuristic algorithm explores stationary points of a surrogate problem obtained from replacing the feasible region of the original formulation with an approximation hypersphere. We also used a spherical relaxation of the original feasible region to draw an upper bound on the optimal solution value of this problem. The exact method employs this bound in a pruning subroutine to reduce the search space. Our computational experiment results show the advantage of the exact method presented in this paper over the branch-and-cut methods proposed for this problem in the literature. It was also shown that the construction heuristic algorithm generates high-quality solutions when the underlying graph is not very sparse.

Acknowledgements We would like to thank Dr. Renata Mansini for providing the test instances, and the anonymous referees, whose feedback helped us to improve the paper.

References

1. Basagni, S.: Finding a maximal weighted independent set in wireless networks. Telecommun. Syst. **18**(1–3), 155–168 (2001)
2. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer Science & Business Media, New York (2011)
3. Brendel, W., Amer, M., Todorovic, S.: Multiobject tracking as maximum weight independent set. In: IEEE conference on computer vision and pattern recognition (CVPR), pp. 1273–1280. IEEE (2011)
4. Butenko, S., Wilhelm, W.: Clique-detection models in computational biochemistry and genomics. Eur. J. Oper. Res. **173**, 1–17 (2006)
5. Carraghan, R., Pardalos, P.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**, 375–382 (1990)
6. Colombi, M., Mansini, R., Savelsbergh, M.: The generalized independent set problem: polyhedral analysis and solution approaches. Eur. J. Oper. Res. **260**(1), 41–55 (2017)
7. Forsythe, G.E., Golub, G.H.: On the stationary values of a second degree polynomial on the unit sphere. SIAM J. Appl. Math. **13**, 1050–1068 (1965)
8. Hochbaum, D.S.: Solving integer programs over monotone inequalities in three variables: a framework for half integrality and good approximations. Eur. J. Oper. Res. **140**(2), 291–321 (2002)
9. Hochbaum, D.S.: 50th anniversary article: selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today. Manag. Sci. **50**(6), 709–723 (2004)
10. Hochbaum, D.S., Pathria, A.: Forest harvesting and minimum cuts: a new approach to handling spatial constraints. For. Sci. **43**(4), 544–554 (1997)
11. Horaud, R., Skordas, T.: Stereo correspondence through feature grouping and maximal cliques. IEEE Transactions on Pattern Analysis and Machine Intelligence **11**(11), 1168–1180 (1989)
12. Hosseinian, S., Fontes, D.B.M.M., Butenko, S.: A quadratic approach to the maximum edge weight clique problem. In: XIII Global optimization workshop, Vol. 16, pp. 125–128 (2016)
13. Hosseinian, S., Fontes, D.B.M.M., Butenko, S.: A nonconvex quadratic optimization approach to the maximum edge weight clique problem. J. Glob. Optim. **72**, 219–240 (2018)
14. Johnson, D.S., Trick, M.A. editors: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society, Providence, RI (1996)
15. Khalil, E.B., Dilkina, B., Nemhauser, G.L., Ahmed, S., Shao, Y.: Learning to run heuristics in tree search. In: Proceedings of the international joint conference on artificial intelligence. AAAI Press, Melbourne (2017)
16. Kochenberger, G., Alidaee, B., Glover, F., Wang, H.: An effective modeling and solution approach for the generalized independent set problem. Optim. Lett. **1**(1), 111–117 (2007)
17. Luenberger, D.G., Ye, Y.: Linear and Nonlinear Programming, third edn. Springer, New York (2008)
18. Sanghavi, S., Shah, D., Willsky, A.S.: Message passing for maximum weight independent set. IEEE Trans. Inf. Theory **55**(11), 4822–4834 (2009)