Comput Optim Appl (2013) 56:113–130
DOI 10.1007/s10589-013-9548-5

Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations

Svyatoslav Trukhanov · Chitra Balasubramaniam · Balabhaskar Balasundaram · Sergiy Butenko

Received: 4 March 2011 / Published online: 14 March 2013

Abstract Given a simple undirected graph, the problem of finding a maximum subset of vertices satisfying a *nontrivial*, *interesting* property Π that is *hereditary on induced subgraphs*, is known to be NP-hard. Many well-known graph properties meet the above conditions, making the problem widely applicable. This paper proposes a general purpose exact algorithmic framework to solve this problem and investigates key algorithm design and implementation issues that are helpful in tailoring the general framework for specific graph properties. The performance of the algorithms so derived for the *maximum s-plex* and the *maximum s-defective clique* problems, which arise in network-based data mining applications, is assessed through a computational study.

Keywords Russian Doll Search · Clique relaxations · s-plex · s-defective clique

Electronic supplementary material The online version of this article (doi:10.1007/s10589-013-9548-5) contains supplementary material, which is available to authorized users.

S. Trukhanov
Microsoft Corp., One Microsoft Way, Redmond, WA 98052, USA
e-mail: svytru@microsoft.com

C. Balasubramaniam · S. Butenko (✉)
Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX 77843, USA
e-mail: butenko@tamu.edu

C. Balasubramaniam
e-mail: bcvaidyanath@tamu.edu

B. Balasundaram
School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK 74078, USA
e-mail: baski@okstate.edu

1 Introduction

Given a simple, undirected graph $G = (V, E)$, let $G[S]$ denote the subgraph induced by a subset of vertices $S \subseteq V$. A graph property Π is said to be *hereditary on induced subgraphs*, if for any $S \subseteq V$ such that $G[S]$ satisfies property Π, the deletion of any subset of vertices in $G[S]$ does not produce a graph violating Π. Property Π is said to be *nontrivial* if it is true for a single-vertex graph and is not satisfied by every graph. A property is said to be *interesting* if there are arbitrarily large graphs satisfying Π. Given a fixed graph property Π, the *the maximum Π problem* seeks to find a largest order subset of vertices inducing a subgraph satisfying property Π. Clearly, the maximum Π problem is meaningful only if Π is nontrivial and interesting, therefore, we will assume that these properties are satisfied for all the considered problems without stating this explicitly. Yannakakis [50] obtained a general complexity result for such properties Π as stated in Theorem 1 (see also [28, 29, 51]).

Theorem 1 [50] *The maximum Π problem for nontrivial, interesting graph properties that are hereditary on induced subgraphs is NP-hard.*

The broad scope of this result is evident from the following examples of Π that meet (what are henceforth referred to as) *the Yannakakis conditions*: clique, independent set, planar, outerplanar, perfect, bipartite, complete bipartite, acyclic, degree-constrained, interval, and chordal graphs among others. In particular, the proof of the above theorem relies heavily on properties of cliques and independent sets, which are defined next. A *clique* $C \subseteq V$ is a subset of vertices such that $G[C]$ is complete, i.e., vertices of C are pairwise adjacent. An *independent set* is a subset of pairwise nonadjacent vertices in a graph. Clearly, C is a clique in G if and only if C is an independent set in the complement graph $\bar{G} = (V, \bar{E})$.

A *maximum* clique (independent set) is a clique (independent set) of the largest cardinality in the graph, whose size is the *clique number* (*independence number*) of G denoted by $\omega(G)$ ($\alpha(G)$). The *maximum clique* and the *maximum independent set* are well-known problems in combinatorial optimization which are NP-hard [22] and hard to approximate [24]. Weighted versions of these problems seek to find a maximum weight clique (independent set) in $G = (V, E)$ where weights $w(v)$, $v \in V$ are assumed to be positive. The weight of a clique (independent set) is the sum of the weights of vertices in the corresponding set. Note that a maximum weight clique (independent set) need not be a maximum size clique (independent set) in the graph, but it will be maximal by inclusion since the weights are positive. An exact algorithm for the weighted problem can solve the unweighted problem (with unit weights), while the converse is not true. Cliques and independent sets have found applications in several areas such as coding theory [15, 42–44], telecommunication [1, 26, 38], fault-diagnosis [23], biochemistry [16], portfolio selection [11, 12], and social network analysis [39] among others. For a survey of the maximum clique problem formulations, exact and heuristic algorithms, see [13].

Motivated by the wide applicability of Yannakakis theorem and the effectiveness of algorithms that can be classified as *Russian Doll Search* (RDS) approaches to the maximum clique problem [17, 35], this paper introduces a general algorithmic

framework for solving the maximum weight Π problem to optimality for any graph property Π that meets the Yannakakis conditions in Sect. 2. This general framework argues for the necessity and sufficiency of Yannakakis conditions in admitting the extension of RDS to graph properties other than cliques. Section 3 discusses algorithm design and implementation issues that help tailor the general framework for specific graph properties, namely, s-plexes [40] and s-defective cliques [52]. These are two hereditary clique relaxations used in graph-based data mining applications, specifically in social and biological network analysis. The performance of the algorithms so derived for *the maximum s-plex problem*, and *the maximum s-defective clique problem*, is assessed through a computational study detailed in Sect. 4. The paper is concluded in Sect. 5, and the Appendix (Online Supplement) includes the complete set of numerical results.

2 The maximum weight Π problem and a general algorithm

Exact combinatorial algorithms for the maximum clique problem have been developed by Bron and Kerbosch [14], Balas and Yu [6], Applegate and Johnson [2], Carraghan and Pardalos [17], Babel [3], Balas and Xue [5], Wood [49], Sewell [41], Östergård [35] and Tomita and Kameda [45]. Östergård also presented an extension designed to solve the weighted version of the problem in [34]. The algorithmic framework presented in this section is a direct generalization of Östergård's algorithm for the maximum weight clique problem [34], which is itself a variant of the RDS algorithm proposed in [47] for constraint satisfaction problems. RDS has recently been applied to other discrete optimization problems such as the Steiner triple covering problem and the maximum transitive subtournament problem [36, 46].

Our choice of the RDS approach as the basis for a general framework applicable to any hereditary property is motivated by its simplicity and effectiveness. It should be noted that a variant of the Carraghan-Pardalos algorithm was used as a benchmark exact algorithm [2] for the maximum clique problem in the Second DIMACS Implementation Challenge [25]. Östergård's algorithm enhances the framework proposed by Carraghan and Pardalos, yielding what is known to be one the fastest exact combinatorial algorithms for solving the maximum clique problem [35]. According to the results reported by Tomita and Kameda [45] for randomly generated test instances and DIMACS benchmark instances [19], their algorithm appears to be the fastest available algorithm in published literature with Östergård's algorithm following closely behind at the time.

Consider a simple, undirected graph $G = (V, E)$, positive weights $w(v)$ for each $v \in V$, and a property Π that satisfies the Yannakakis conditions. Define the invariant $\mu(G)$ as follows:

$$\mu(G) = \max\{w(P) : \ P \subseteq V, \ G[P] \text{ satisfies } \Pi\},$$

where $w(P) = \sum_{v \in P} w(v)$. Finding P^* such that $\mu(G) = w(P^*)$ constitutes the *maximum weight Π problem*. When $w(v) = 1 \ \forall v \in V$, this is the *maximum Π problem*. Note that the assumption of positive weights is not restrictive when Π is hereditary, as vertices with nonpositive weights can be deleted without changing $\mu(\cdot)$. Under this assumption, an optimal P^* such that $w(P^*) = \mu(G)$ will be maximal by

Algorithm 1 RDS Algorithm for The Maximum Π Problem

```
1:  procedure RDS-ALGORITHM(G, Π)
2:      Order(V)                                          ▷ Vertex ordering point
3:      max := 0
4:      for i := n downto 1 do
5:          C := {v ∈ S_i \ {v_i} : {v, v_i} satisfies Π}   ▷ Π-verification
6:          FINDMAXΠ(C, {v_i})
7:          μ(i) := max
8:      end for
9:      return max
10: end procedure

11: procedure FINDMAXΠ(C,P)
12:     if C = ∅ then
13:         if w(P) > max then
14:             max := w(P)
15:         end if
16:         return
17:     end if
18:     while C ≠ ∅ do
19:         if w(C) + w(P) < max then
20:             return                                    ▷ Prune point 1
21:         end if
22:         i := min{j : v_j ∈ C}
23:         if μ(i) + w(P) < max then
24:             return                                    ▷ Prune point 2
25:         end if
26:         C := C \ {v_i}
27:         P' := P ∪ {v_i}
28:         C' := {v ∈ C : P' ∪ {v} satisfies Π}           ▷ Π-verification
29:         FINDMAXΠ(C',P')
30:     end while
31: end procedure
```

inclusion. Based on Theorem 1 and the fact that the unweighted version is a special case, the maximum weight Π problem is also NP-hard.

Algorithm 1 presenting the overall framework proceeds as follows. First, we impose an ordering of vertices in V and use the notation $V = (v_1, v_2, \ldots, v_n)$ to refer to the set of ordered vertices. With the sets $S_i \subseteq V$ defined as $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$, Algorithm 1 computes $\mu(G[S_i])$, denoted simply by $\mu(i)$. Obviously, $\mu(n) = w(v_n)$ and $\mu(1) = \mu(G)$. The algorithm computes the value of $\mu(i)$ starting from $\mu(n)$ and down to $\mu(1)$, and clearly $\mu(i) \geq \mu(i+1)$. Moreover, if $\mu(i) > \mu(i+1)$, then any solution that yields the value $\mu(i)$ contains vertex v_i. Otherwise, there exists a solution P_i, such that $\mu(i) = w(P_i)$ and $v_i \notin P_i$, then $P_i \subseteq S_i \setminus \{v_i\} = S_{i+1}$ implying, $\mu(i+1) = \mu(i)$, a contradiction. Therefore, for the unweighted case with

$w(v_i) = 1, i = 1, \ldots, n$, the above argument implies that,

$$\mu(i) = \begin{cases} \mu(i+1) + 1, & \text{if every corresponding solution contains } v_i \\ \mu(i+1), & \text{otherwise} \end{cases}$$

For the general weighted case, it is important to observe that a similar claim, *i.e.* $\mu(i) = \mu(i+1)$ or $\mu(i) = \mu(i+1) + w(v_i)$ would be *incorrect*. However, $\mu(i) > \mu(i+1)$ implies that v_i belongs to every optimal solution, and $\mu(i) \leq \mu(i+1) + w(v_i)$.

In order to understand the importance of requiring Π to be hereditary on induced subgraphs, consider the following definition of Π which is *not* hereditary on induced subgraphs. G is said to satisfy Π if the diameter of G is at most two (the subset of vertices inducing a subgraph satisfying such a Π is also called a 2-club [9]). Now consider the graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_1, v_i) : i = 2, \ldots, n\}$, *i.e.*, G is a *star graph* with central vertex v_1 and leaves $v_2, \ldots, v_n$. Assuming unit weights, $\mu(i) = 1\ \forall i = 2, \ldots, n$, but $\mu(1) = n$. This is because G satisfies Π while any induced subgraph with at least two vertices obtained by deleting v_1 violates Π.

Algorithm 1 computes $\mu(n), \mu(n-1), \ldots, \mu(1)$ using backtracking, and employs two pruning strategies (discussed later). In particular, the second pruning strategy is based on the *bounded increase* going from $i+1$ to i, which only holds in general (for any graph and any ordering) when Π is hereditary. The second pruning strategy is a key factor contributing to the effectiveness of Östergård's algorithm, which is dependent on the hereditary nature of Π.

The main procedure RDS-ALGORITHM accepts graph G (and property Π) as input and calls the recursive procedure FINDMAXΠ to compute the values of $\mu(i)$. The procedure FINDMAXΠ is the core of the algorithm and finds a maximum weight subgraph with property Π using two sets as the input: (i) the *working set* C (also known as *candidate list* or *candidate set*) is the set of vertices that may be used to build a subgraph with property Π, and (ii) the set of vertices P that represents the currently found subgraph with property Π. The procedure chooses vertices from C one by one, adds a chosen vertex to the current graph P, updates the candidate list, and calls itself with new values of input sets. Obviously, if the candidate set is empty, the procedure terminates returning the best weight value found. Other points of termination are the prune points at Line 20 and Line 24 indicated in Algorithm 1, referred to as type 1 and type 2 pruning, respectively. Pruning of the first type occurs when the weight of the current subgraph together with the weight of the whole candidate set is less than the incumbent *max*. The second type of pruning occurs when the weight of the current subgraph together with $\mu(i)$ (which is best possible in $G[S_i]$) is less than the incumbent *max*.

This general framework to solve the maximum weight Π problem would be a depth-first search order complete enumeration in the absence of both prune points 1 and 2; with only prune point 1, the algorithm is a depth-first search order implicit enumeration. The latter results in a "Carraghan-Pardalos type" backtracking algorithm for the maximum clique problem [17]. The second type of pruning is facilitated by the vertex ordering, and the ensuing arguments on bounded increase in $\mu(i)$ developed by Östergård [34, 35] for the maximum clique problem. One of the key design

elements that heavily influences the effectiveness of this algorithm is the choice of a vertex ordering scheme that "encourages" type 2 pruning. While the general framework itself conceptually subsumes other approaches, key implementation issues need to be addressed to make the resulting algorithm as effective for the particular problem under consideration as the Östergård's algorithm is for the maximum clique problem. In the remainder of this paper, we study the development of such an implementation for *the maximum s-plex problem* and *the maximum s-defective clique problem*.

3 Adaptation of the algorithm for two hereditary clique relaxations

Definition 1 Given a simple, undirected graph $G = (V, E)$ and a fixed positive integer s, a subset of vertices S is an s-plex if it satisfies the following property:

$$\left|N(v) \cap S\right| \geq |S| - s \quad \forall v \in S,$$

where $N(v)$ is the neighborhood of v (i.e., $N(v) = \{u \in V : (u, v) \in E\}$). In other words, the minimum vertex degree in $G[S]$ is at least $|S| - s$.

The case $s = 1$ corresponds to a clique, and for $s > 1$, the s-plex model is a relaxation of clique which allows for at most $s - 1$ *non-neighbors* for each vertex.

Definition 2 Given a simple, undirected graph $G = (V, E)$ and a fixed nonnegative integer s, an s-defective clique S is a subset of vertices that satisfies the following property:

$$\left|E[S]\right| \geq \binom{|S|}{2} - s,$$

where $E[S]$ is the edge set of the subgraph induced by S.

A 0-defective clique is a clique, and the s-defective clique model for $s > 0$ relaxes the clique requirement of having all possible edges by allowing at most s edges to be absent from the induced subgraph.

The maximum weight s-plex problem and the maximum weight s-defective clique problem are the optimization problems of interest. The s-plex model was introduced by Seidman and Foster [40] to represent *cohesive subgroups* in *social network analysis* [48]. A typical social network is the *acquaintance network* were vertices represent people and an edge indicates that the two people represented by the end-points know each other. Seidman and Foster were motivated by the observation that cliques were capable of modeling "ideal" cohesive subgroups in which every individual knew everyone else, but were not necessarily suitable for detecting real-life cohesive subgroups. Polyhedral approaches [8, 31] and exact combinatorial algorithms [32, 33] have been developed to solve the maximum s-plex problem. In particular, McClosky and Hicks [32] independently extended Carraghan-Pardalos [17] and Östergård [35] algorithms for the maximum s-plex problem. This paper demonstrates improvements over the results of McClosky and Hicks [32] and Balasundaram et al. [8] in Sect. 4 that are achieved via improved algorithm design and implementation.

The s-defective clique model was introduced by Yu et al. [52], who use it to represent clusters in protein interaction networks. In such networks, the proteins in an organism are represented by vertices, and an edge indicates that the proteins corresponding to its end-points are known to interact based on biological experiments. These biological experiments can be high-throughput and noisy [4, 52] resulting in large-scale network models representing an erroneous edge set. This motivated Yu et al. [52] to use s-defective cliques as an alternative to cliques to facilitate the detection of *protein complexes* in the presence of missing edges. However, a heuristic method is proposed therein and no exact algorithms are currently available for this problem. This paper investigates the performance of the proposed general-purpose exact algorithm when tailored to this problem.

It should be noted that in addition to s-plexes and s-defective cliques several other clique relaxations were introduced and studied in the context of graph-based data mining and clustering applications [10, 37, 39], however, unlike s-plexes and s-defective cliques, they do not satisfy the Yannakakis conditions. Tailoring the general framework of Algorithm 1 for the maximum s-plex problem and the maximum s-defective clique problem requires us to focus our attention on two key points identified in the algorithm, (i) vertex ordering, which significantly affects type 2 pruning (also noted by Östergård [35], McClosky and Hicks [32]), and (ii) Π-verification, a step that is invoked very frequently and hence, can be a potential source for performance improvement. These issues are the focus of the investigation in the remainder of this paper.

3.1 Candidate set generation and Π-verification

Recall that Algorithm 1 makes recursive calls with a candidate set C and a current solution P. The candidate set has the property that $\{v\} \cup P$ satisfies Π for each $v \in C$. The first key challenge with respect to algorithm design when tailoring Algorithm 1 for a specific graph property Π is the candidate set update procedure that verifies and, hence, maintains Π. For cliques, the candidate set is just the intersection of neighborhoods of vertices from the current clique. If v_i was added at current iteration, then the candidate set C' for the next iteration is the intersection of the current candidate set and the neighborhood of the newly added vertex: $C' = C \cap N(v_i)$.

However, in the case of s-plex and s-defective clique, the definition permits a limited number of non-neighbors. The new candidate set must be generated by explicitly verifying and selecting each vertex from the current candidate set that forms an s-plex/s-defective clique when added to the current s-plex/s-defective clique. This forces the use of Π-verification procedures such as Algorithm 2 and Algorithm 3. The function is called for each member of the current candidate set each time the candidate set needs to be updated. The speed of the Π-verification procedure determines the speed of candidate set update process, and significantly influences the overall algorithm running time, given the number of such calls. We need to focus on improving the running time of this procedure, as well as reducing the number of calls to this procedure. This subsection focuses on the former while Sect. 3.3 focuses on the latter.

Algorithms 2 and 3 present simple verification procedures for s-plex and s-defective clique, respectively. The function IsPlex1 has a quadratic running time

Algorithm 2 s-Plex verification procedure

```
1: function IsPlex1(K, s)
2:     for v ∈ K do
3:         if deg_{G[K]}(v) < |K| − s then
4:             return false
5:         end if
6:     end for
7:     return true
8: end function
```

Algorithm 3 s-Defective clique verification procedure

```
1: function IsDefClique(K, s)
2:     count ← 0
3:     for v ∈ K do
4:         count ← count + |K| − 1 − deg_{G[K]}(v)
5:         if count > 2s then
6:             return false
7:         end if
8:     end for
9:     return true
10: end function
```

complexity with respect to the size of its argument K with a linear-time procedure for determining the induced degree in each iteration of the for-loop. We can improve this procedure by using additional information available from the previous steps. Assume that the vertex u has just been added to the s-plex K in the current iteration, so that the new s-plex is $K' = K \cup \{u\}$. We know that $K \cup \{u\}$ is an s-plex and $K \cup \{v\}$ is an s-plex for any vertex v from C. To determine whether v will be included in the new candidate set C', we need to verify that $K \cup \{u\} \cup \{v\}$ is an s-plex. Let us call a vertex $i \in K$ *saturated* if $|K \cap N(i)| = |K| - s$, which means it is not possible to add vertices not in $N(i)$ to K without violating the degree requirement. Since $K \cup \{v\}$ is already known to be an s-plex, the only possible violations may be caused by the newly added vertex u. First, addition of u could increase the number of non-neighbors for some vertex $y \in K \setminus N(u)$ by one. If such a vertex y becomes saturated in K', then C' must be a subset of $N(y)$. Second, the vertex u itself could be saturated in K', in which case $C' \subseteq N(u)$. Unsaturated vertices in K' do not create any restrictions for the candidate set. Note that if $s = 1$, then every vertex in K is saturated and these observations unify to yield the intersection of neighborhoods. These considerations allow one to create a faster procedure for updating the candidate set. After adding a new vertex we determine the list of saturated vertices. Then the new candidate set is obtained by intersecting the current candidate set with the neighborhood of each saturated vertex. In order to quickly identify vertices that become saturated, we keep and update the list and number of non-neighbors for vertices in K. Algorithm 4 formalizes the resulting *incremental* s-plex verification procedure.

Algorithm 4 Incremental s-plex verification procedure

```
1:  function MakeSaturatedList(K, s)
2:      u ← last vertex added to K
3:      S ← ∅                                          ▷ Saturated vertex list
4:      for v ∈ K \ {u} do
5:          if (u, v) ∉ E(G) then
6:              nncnt[u] ← nncnt[u] + 1    ▷ update number of non-neighbors for u
7:              nncnt[v] ← nncnt[v] + 1    ▷ update number of non-neighbors for v
8:              if nncnt[v] = s − 1 then
9:                  S ← S ∪ {v}
10:             end if
11:         end if
12:     end for
13:     if nncnt[u] = s − 1 then
14:         S ← S ∪ {u}
15:     end if
16:     return S
17: end function

18: function IsPlex2(K, S, v)
19:     for u ∈ S do
20:         if (u, v) ∉ E(G) then
21:             return false
22:         end if
23:     end for
24:     return true
25: end function
```

Recall that the naive s-plex verification procedure runs in $O(|K|^2)$ time. The new procedure consists of two parts: generation of the list of saturated vertices, which can be done in $O(|K|)$ time and is performed once; and verification whether a vertex is in the neighborhood of saturated vertices, which is performed for every vertex from the candidate list. The theoretical complexity of the second part is $O(|K|)$, but each vertex can be a member of the saturated vertex list only once during the whole process of building K (once a vertex is included in this list, the candidate list will be intersected with this vertex neighborhood, and no more vertices from the candidate list may satisfy the condition in line 5 of Algorithm 4). As will be discussed later, the comparison of running time of the maximum weight s-plex algorithms that utilize the original and the alternative s-plex verification procedures shows a significant improvement in the performance by using the latter approach, especially for larger s.

Since an s-defective clique is also an $(s+1)$-plex, Algorithm 4 is valid for the maximum s-defective clique problem as well. Algorithm 4 verifies if the vertex subset K is an $(s+1)$-plex and if so, calls Algorithm 3 for checking if K is an s-defective clique. This reduces the number of calls for Algorithm 3, which runs in $O(|K|^2)$ time.

3.2 Scale-reduction based on 2-neighborhood for the unweighted case

The simple fact that for any vertex v from clique C, $C \subseteq N[v]$, allows us to reduce the problem of finding a maximum clique from the whole graph to several single-vertex neighborhoods. Seidman and Foster [40] showed that if G is an s-plex with more than $2s - 2$ vertices, then the diameter of G is at most two. The connectivity of large s-plexes has already been used indirectly in the previous subsection, where the candidate list was shrunk by intersection of neighborhoods of saturated vertices, which means that all other candidates to be included in s-plex are connected to the saturated vertex. This observation will also be used heuristically to order vertices based on the size of their distance-two neighborhoods in Sect. 3.3.2. Unlike with cliques, the containment of solution inside the (double) neighborhood only holds for s-plexes of size at least $2s - 1$ and may not hold in general for smaller order s-plexes. At first glance, the bounded diameter property cannot be used in the algorithm, since in a weighted graph there can be an s-plex of cardinality less than $2s - 1$, but of a weight larger than that of any s-plex of size greater than $2s - 2$. However, we can still modify the original maximum s-plex algorithm to improve its performance for instances of the problem containing s-plexes of a certain size in the unweighted case as follows. After adding a new vertex v to the formed s-plex, the working set outside of $N_2[v]$ is simply cut off, thus reducing the problem of finding the required s-plex to $N_2[v]$. Then, if the resulting solution has cardinality at least $2s - 1$, this solution must be a maximum s-plex in G. Otherwise, if the resulting s-plex has cardinality smaller than $2s - 1$, then the original algorithm must be executed in order to find an optimal solution.

3.3 Vertex ordering

While Sect. 3.1 dealt with improving the s-plex and the s-defective clique verification procedure, this subsection investigates approaches that aim to decrease the number of calls to this procedure. Algorithm 1 is a branch and bound algorithm, where Π-verification procedure is called in each branching node multiple times. Given the intractability of the problem, it is unlikely to be able to provably reduce the number of such calls in general. The number of calls would be reduced if, (1) we can shrink the working set faster (*i.e.*, the branch is cut off because of feasibility), and (2) we can prune more often (*i.e.*, the branch is cut off because of the bound). Both these goals are influenced by the ordering of vertices in line 2 prior to commencing the main iterations in line 4 of Algorithm 1. The following subsections discuss different vertex ordering schemes for weighted and unweighted graphs.

3.3.1 Weight-based ordering

Since we seek an s-plex/s-defective clique of maximum weight, a weight-based ordering is an alternative. Recall that the algorithm processes the vertices starting from v_n down to v_1. A greedy approach would suggest a non-increasing order of vertex weights *i.e.*, an ordering such that $w(v_n) \geq w(v_{n-1}) \cdots \geq w(v_1)$. However, in this case the values $\mu(i)$ and the weight of the incumbent s-plex/s-defective clique, *max*,

grow quickly in the beginning, but subsequently, as *max* is already close, or, perhaps, equal, to the optimal value, the growth stalls. Thus, the values of $\mu(i)$ and *max* remain equal for a long time, and the algorithm proceeds without any improvement. This leads to the situation where the pruning in line 24 of Algorithm 1 does not occur.

The smallest-to-the-largest-weight ordering benefits from the opposite effect: the values of $\mu(i)$ are maintained to be as small as possible in order to yield more type 2 pruning points. Therefore, ordering the vertices in nondecreasing order of their weights provides a reasonable weight-based ordering. It should be noted that the Östergård's algorithm for the maximum weight clique used a similar ordering with additional tie-breaking rules when several vertices have the same weight. In our study, we break the ties by simply ordering vertices according to their original labels. In particular, for instances with all equal weights and for unweighted instances this results in the natural ordering from vertex labels, which we call the *control ordering*.

3.3.2 Degree-based ordering

When searching for a maximum clique in an unweighted graph, it is natural to order the vertices according to their degree. The ordering procedure does start with the minimum degree vertex v_n, however the further order of vertices depends on their degrees in the subgraph induced by the subset of vertices that have not yet been ordered and hence the resulting order is not the same as the one resulting from sorting the vertices by their degree. Such a degree-based ordering was used in the Carraghan-Pardalos algorithm for the maximum clique problem, and is supported by the fact that the clique C that contains vertex v is a subset of $N[v]$, so starting with vertices of smaller degree provides small values for $\mu(i)$, which can yield higher success rate with the type 2 prune points (Algorithm 1, line 24).

3.3.3 2-Neighborhood-based ordering

Since an s-plex containing v_i is not necessarily a subset of $N[v_i]$, the ordering is not as clearly justified as for the maximum clique algorithm, even though the number of non-neighbors is limited to $s-1$. The same holds for s-defective clique. Hence, the degree-based ordering for the maximum clique problem is modified to suit the properties of the problems in hand. The result of Seidman and Foster [40] is helpful in this regard. If the unknown optimal s-plex K^* is sufficiently large, *i.e.,* $|K^*| > 2s-2$, then every maximum s-plex has diameter at most two. So any maximum s-plex containing an arbitrary vertex v is a subset of $N_G^2[v] = \{u \in V(G) : d_G(v,u) \leq 2\}$. This simple modification provides a new ordering based on the size of the 2-neighborhood of a vertex instead of its degree. The same considerations apply to the maximum s-defective clique problem.

3.3.4 Coloring based ordering

Another popular vertex ordering for the maximum clique problem emphasizes reducing the working set as fast as possible and is based on the fact that no more than one vertex from an independent set can be included in a clique. The corresponding ordering partitions a graph into independent sets (performs graph coloring) and groups

vertices from the same color class together. Since the problem of graph coloring is NP-hard, the coloring is performed greedily, and the corresponding ordering is based on the order in which colors were assigned. Such an ordering is expected to reduce the working set fast, since all vertices of some color will be removed after one vertex of this color is chosen for inclusion into clique. Vertices can be ordered starting from the color class of the largest cardinality or in the opposite order.

Coloring guarantees that only one vertex of each color could be included in a clique. However, this is not the case for s-plex, as any s vertices form an s-plex by definition, and the implementation of coloring-based ordering must be modified accordingly. Thus, we replace partitioning into independent sets (classical coloring) by partitioning into independent set relaxations. One such structure is the *co-s-plex*, defined as a subset $S \subseteq V$ of vertices satisfying the following property:

$$\left|N(v) \cap S\right| \leq s - 1 \quad \forall v \in S.$$

Clearly, a co-s-plex is an independent set for $s = 1$ and its relaxation for $s > 1$. Further, $S \subseteq V$ is a co-s-plex in G if and only if S is an s-plex in the complement of G.

Balasundaram et al. [8] showed that at most $2s - 2 + (s \mod 2)$ vertices from a co-s-plex may be included into an s-plex. It is not known how many vertices from a co-$\bar{s}$-plex may be included in an s-plex for arbitrary values of $\bar{s}$ and s, but obviously for a fixed s this number increases when $\bar{s}$ increases. On the one hand, increasing $\bar{s}$ creates more branches at each step, however, on the other hand, the number of partitions in the graph will be smaller. The problem of partitioning a graph into co-s-plexes is known in the literature as defective coloring [18, 21] and is NP-hard, so we use a simple greedy procedure (Algorithm 5) to obtain the needed partitions. If the graph is weighted then the next vertex in line 6 of Algorithm 5 is chosen as the smallest weight non-colored vertex. For vertices with the same weight, the one with the largest neighborhood weight is chosen, following the rules similar to [34, 35].

Experiments were conducted for $\bar{s} = 1, 2, \ldots, 5$ using both ordering strategies (largest color class first and the opposite) with unweighted instances of the maximum s-plex problem and the maximum $(s - 1)$-defective clique problem. The main observation from these experiments is that the ordering based on defective coloring typically yields better algorithm performance than the control ordering. Again, there is no conclusive evidence as to which coloring provides the best results. In many cases, the classical coloring (i.e., partitioning the graph into independent sets) outperforms the other considered orderings, but quite often the best result is obtained when $\bar{s} = s$. Less often, the best running time is given by values of $\bar{s}$ that are in between 1 and s, and even more rarely, when $\bar{s} > s$.

The vertex ordering techniques considered in this section are heuristic in nature and cannot provide a predictable performance improvement on all problem instances, as was in the case of candidate set generation in Sect. 3.1. But when they do provide an improvement, it is usually much more significant than that obtained using the techniques from Sect. 3.1 alone.

Algorithm 5 Defective $\bar{s}$-coloring based vertex ordering

```
1: procedure OrderColoring(V, s̄)
2:     U ← ∅                                          ▷ Set U is ordered
3:     col[v] ← 0 ∀v ∈ V                              ▷ Color assigned to vertex
4:     colnum[v] ← 0 ∀v ∈ V                           ▷ Number of neighbors of the same color
5:     for i = 1 to |V| do                            ▷ Assign the colors
6:         u ← min{v : col[v] = 0 and v ∈ V}
7:         C ← {col[v] : colnum[v] < s̄ − 1 ∀v ∈ N(u)}
8:         if C = ∅ then
9:             col[u] ← max col[v] + 1
10:        else
11:            col[u] ← min C
12:        end if
13:        colnum[u] = |{v ∈ N(u) : col[u] = col[v]}|
14:        colnum[v] = colnum[v] + 1 ∀v ∈ N(u) : col[u] = col[v]
15:    end for
16:    for j = 1 to max{col[v] : v ∈ V} do
17:        U ← U ⊕ {u : col[u] = j}                   ▷ ⊕ means "append to the end"
18:    end for
19:    return U
20: end procedure
```

4 Results of numerical experiments

All numerical experiments presented in this paper were conducted on DELL OPTIPLEX GX620 computer with INTEL(R) CORE(TM) 2 QUAD 3 GHz processor and 4 GB of RAM. The algorithm was implemented in the C programming language, using MICROSOFT VISUAL C++ .NET 2010 (V 10.0) development environment for Win32 platform. Despite the dual core feature of the CPU, the algorithm is implemented using single threaded mode and cannot use this hardware advantage.

In the experiments reported in this paper, we considered only unweighted instances of the problems of interest. A set of preliminary experiments was conducted to facilitate the choice of algorithm settings used in the set of tests reported below. In particular, in the reported experiments the algorithm always used the incremental s-plex verification routine, as described in Sect. 3.1 and the diameter based pruning from Sect. 3.2. Degree-based vertex ordering described in Sect. 3.3.2, which exhibited the best overall performance for both problems in the preliminary tests, is considered for the reported experiments. It should be noted that this ordering is the same as the one used in Carraghan-Pardalos algorithm for the maximum clique problem [17]. Therefore, we will refer to this ordering as the *standard ordering*.

4.1 Standard testbed

The standard testbed of instances used in our experiments consists of two groups. The first group is comprised of benchmark clique instances from the Second

Table 1 Running time comparison with algorithms of Balasundaram et al.

Graph	Runtime, sec		
	BC-MIS	BC-C2PLX	Algorithm 1
MANN_a27.clq	>10800	>10800	>10800
MANN_a9.clq	0.262	0.289	0.02
brock200_1.clq	>10800	>10800	>10800
c-fat200-1.clq	25.891	212.239	0.13
c-fat200-2.clq	24.235	7636.49	0.05
c-fat500-1.clq	1263.81	9587.21	0.06
c-fat500-10.clq	>10800	>10800	0.11
c-fat500-2.clq	2985.04	>10800	0.08
c-fat500-5.clq	10142.8	>10800	0.16
hamming6-2.clq	0.421	1.686	0.03
hamming6-4.clq	4.609	6.767	0.02
hamming8-2.clq	>10800	>10800	0.14
hamming8-4.clq	>10800	>10800	4.76
johnson16-2-4.clq	>10800	>10800	3311.91
johnson8-2-4.clq	1.952	1.171	0.01
johnson8-4-4.clq	1951.87	3283.15	0.03
keller4.clq	>10800	>10800	460.01

DIMACS Challenge [19, 25]. The second group is comprised of instances from the Tenth DIMACS Challenge [20] and Standford Large Network Database Collection (SNAP) [27]. The instances in this group are typically very large and sparse, and hence a reduction on the graph size using the so-called *peeling procedure* originally proposed for the maximum clique problem [1], which, given a lower bound $|S|$ on the clique number (found, e.g., heuristically) recursively removes vertices of degree less than $|S| - s$. Obviously, the peeling procedure cannot remove any vertices belonging to a maximum s-plex or s-defective clique. The total number of instances considered is 63, and we preserve their original names in Tables 1–2, 3–6.

Tables 3 and 4 in the Appendix (Online Supplement) show the parameters of the instances, including the graph order, size, density, clique number, s-plex number for $s = 2, 3, 4, 5$ and s-defective clique number for $s = 1, 2, 3, 4$. The clique numbers for DIMACS graphs were obtained from [19] and other sources. The s-plex numbers and the s-defective clique numbers were obtained by our algorithm. If the algorithm could not find the optimal solution within a 3-hour time limit, the size of the incumbent s-plex/s-defective clique is provided, indicated with "$\geq$" in the tables.

Tables 5 and 6 in the Appendix (Online Supplement) provide the running time of the maximum weight s-plex and s-defective clique algorithm in CPU seconds. In these tables, the column labeled by "Reduced" specifies the number of vertices remaining in the graph after applying the peeling procedure, with an asterisk used to mark the cases where no reduction occurred.

As mentioned before, the ordering scheme considered was the standard ordering due to its superior performance on most instances. However, it should be noted that for several instances other ordering schemes explained in Sect. 3.3 performed much

Table 2 Running time comparison with McClosky-Hicks algorithm

Graph	Runtime, sec					
	$s=2$		$s=3$		$s=4$	
	MH	Alg. 1	MH	Alg. 1	MH	Alg. 1
brock200_2	64	6.55	>3600	446.68	>3600	>3600
brock200_4	>3600	398.65	>3600	>3600	>3600	>3600
c-fat200-1	0	0.13	0	0.08	18	0.05
c-fat200-2	0	0.05	0	0.03	3	0.05
c-fat500-1	0	0.06	8	0.11	1234	0.11
c-fat500-2	0	0.08	2	0.06	92	0.09
c-fat500-5	0	0.16	1	0.2	8	0.64
c-fat500-10	0	0.11	0	0.09	4	0.2
hamming6-2	0	0.03	1	0.19	951	178.24
hamming6-4	0	0.02	0	0.05	1	0.22
hamming8-2	1	0.14	>3600	>3600	>3600	>3600
hamming8-4	58	4.76	>3600	>3600	>3600	>3600
johnson8-2-4	0	0.01	0	0.01	0	0.05
johnson8-4-4	0	0.03	35	6.41	>3600	906.06
johnson16-2-4	>3600	3311.91	>3600	>3600	>3600	>3600
keller4	913	460.01	>3600	>3600	>3600	>3600
MANN_a9	0	0.02	2	0.03	141	4.18
MANN_a27	>3600	>3600	>3600	0.24	>3600	>3600
p_hat300_1	5	0.48	416	41.74	>3600	3041.44
p_hat300_2	>3600	992.36	>3600	>3600	>3600	>3600

better than the standard ordering. Most notably, for the instance c-fat200-5.clq the running times obtained using the ordering by the smallest degree first for $s = 2, 3$, coloring-based ordering with the smallest color class first for $s = 4$, and the ordering that uses the smallest double neighbourhood size first for $s = 5$, were below 1 sec, whereas the run using the standard ordering failed to terminate after 3 hours.

4.2 Comparison with existing approaches

Since the recent introduction of the maximum s-plex problem to the operations research literature by Balasundaram at el. [7, 8], at least four approaches (including the original one and the current one) for solving the problem have been developed. The first approach was proposed in [7, 8], where a branch-and-cut method for the maximum s-plex problem was developed and tested on a subset of DIMACS graphs. Two versions of the branch-and-cut algorithm for the maximum s-plex problem were implemented. Both versions employ CPLEX for solving the IP formulation of the problem, the difference being the cuts used. One version incorporates the maximum independent set cuts (referred as BC-MIS), while the other incorporates co-s-plex cuts (referred as BC-C2PLX).

Table 1 shows the running time of the three approaches: BC-MIS and BC-C2PLX by Balasundaram et al. [8] and Algorithm 1. In most cases, our algorithm outperforms both branch-and-cut approaches, possibly due to tuning of this algorithm to the particular problem. The branch-and-cut approaches were less tuned to this particular problem as they only employed problem-specific cutting planes while inheriting default CPLEX branching and search strategies.

McClosky and Hicks [30, 32] considered two different algorithms for finding a maximum s-plex in a graph. The first algorithm is based on Carraghan-Pardalos [17] ideas, while the second is based on the aforementioned Östergård's algorithm for cliques. Both algorithms were tested on a subset of DIMACS benchmarks. Since their second algorithm outperforms the first on all test instances, we only compare our results with the second algorithm. Since McClosky and Hicks limited the algorithm execution time to one hour in their experiments, we also disregard all results that were obtained in a larger amount of time. The numerical results were provided for $s = 2, 3, 4$ with precision 1 sec, the running time 0 in McClosky-Hicks experiments was interpreted as <1 sec. The time limit was set to 1 hour in this case since the same termination criterion was used in [32]. Table 2 provides the running times, where the columns marked with "MH" refer to McClosky-Hicks experiments and those marked by "Alg. 1" show our results. From the results, we conclude that our algorithm demonstrated better running time performance, which can be explained by more refined procedures for candidate set generation and pruning points used.

5 Conclusion

This paper exploits the connections between the complexity result for the problem of finding a maximum subset of vertices satisfying a nontrivial, interesting property Π that is hereditary on induced subgraphs, established in 1978 by Yannakakis, and RDS-type algorithms for the maximum clique problem proposed by Carraghan and Pardalos in 1990 and enhanced by Östergård in 2002. We first extend the ideas used in these algorithms to the weighted version of node deletion problems satisfying the conditions of Yannakakis theorem and then perform a problem-specific study of the approach with the maximum s-plex and the maximum s-defective clique problems, which are two hereditary relaxations of the maximum clique problem that are of interest in graph-based data mining applications. The comparison of the proposed algorithm with two other existing methods for the maximum s-plex problem published in the literature on standard testing instances demonstrates the improved performance of the proposed algorithm and its problem-specific enhancements. To the best of our knowledge, our adaptation of the proposed algorithm to the maximum s-defective clique problem is the first reported exact algorithm for this problem. The proposed general algorithmic framework as well as some of the observations made in the process of experimenting with the maximum s-plex and the maximum s-defective clique instances could be used to develop successful implementations for other hereditary structures in graphs.

Acknowledgements We would like to thank the referees for providing useful suggestions, which helped us to significantly improve the paper. This research was supported in part by the US Department of Energy Grant DE-SC0002051 and the US Air Force Office of Scientific Research Grants FA9550-09-1-0154 and FA9550-12-1-0103.

References

1. Abello, J., Pardalos, P.M., Resende, M.G.C.: On maximum clique problems in very large graphs. In: Abello, J., Vitter, J. (eds.) External Memory Algorithms and Visualization. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 50, pp. 119–130. American Mathematical Society, Providence (1999)
2. Applegate, D., Johnson, D.S.: dfmax.c [c program, second dimacs implementation challenge]. http://dimacs.rutgers.edu/pub/challenge/graph/solvers/
3. Babel, L.: Finding maximum cliques in arbitrary and in special graphs. Computing **46**(4), 321–341 (1991)
4. Bader, J.S., Chaudhuri, A., Rothberg, J.M., Chant, J.: Gaining confidence in high-throughput protein interaction networks. Nat. Biotechnol. **22**(1), 78–85 (2004)
5. Balas, E., Xue, J.: Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. Algorithmica **15**, 397–412 (1996)
6. Balas, E., Yu, C.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**, 1054–1068 (1986)
7. Balasundaram, B.: Graph theoretic generalizations of clique: optimization and extensions. PhD thesis, Texas A&M University, College Station, Texas, USA (2007)
8. Balasundaram, B., Butenko, S., Hicks, I.V.: Clique relaxations in social network analysis: the maximum k-plex problem. Oper. Res. **59**(1), 133–142 (2011)
9. Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. J. Comb. Optim. **10**(1), 23–39 (2005)
10. Balasundaram, B., Mahdavi Pajouh, F.: Graph theoretic clique relaxations and applications. In: Pardalos, P.M., Du, D.-Z., Graham, R. (eds.) Handbook of Combinatorial Optimization, 2nd edn. Springer, Berlin (2013). doi:10.1007/978-1-4419-7997-1_9
11. Boginski, V., Butenko, S., Pardalos, P.: Mining market data: a network approach. Comput. Oper. Res. **33**, 3171–3184 (2006)
12. Boginski, V., Butenko, S., Pardalos, P.M.: On structural properties of the market graph. In: Nagurney, A. (ed.) Innovation in Financial and Economic Networks. Edward Elgar, London (2003)
13. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, pp. 1–74. Kluwer Academic, Dordrecht (1999)
14. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques on an undirected graph. Commun. ACM **16**, 575–577 (1973)
15. Brouwer, A., Shearer, J., Sloane, N., Smith, W.: A new table of constant weight codes. IEEE Trans. Inf. Theory **36**, 1334–1380 (1990)
16. Butenko, S., Wilhelm, W.: Clique-detection models in computational biochemistry and genomics. Eur. J. Oper. Res. **173**, 1–17 (2006)
17. Carraghan, R., Pardalos, P.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**, 375–382 (1990)
18. Cowen, L., Goddard, W., Jesurum, C.E.: Defective coloring revisited. J. Graph Theory **24**(3), 205–219 (1997)
19. DIMACS. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge (1995). Online: http://dimacs.rutgers.edu/Challenges/. Accessed March 2007
20. DIMACS. Graph partitioning and graph clustering: tenth Dimacs implementation challenge (2011). Online: http://www.cc.gatech.edu/dimacs10/index.shtml. Accessed July 2012
21. Frik, M.: A survey of (m, k)-colorings. In: Gimbel, J., Kennedy, J.W., Quintas, L.V. (eds.) Quo Vadis, Graph Theory? Annals of Discrete Mathematics, vol. 55, pp. 45–58. Elsevier, New York (1993)
22. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
23. Hasselberg, J., Pardalos, P.M., Vairaktarakis, G.: Test case generators and computational results for the maximum clique problem. J. Glob. Optim. **3**(4), 463–482 (1993)

24. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math. **182**, 105–142 (1999)
25. Johnson, D.S., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiablility: Second Dimacs Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society, Providence (1996)
26. Krishna, P., Chatterjee, M., Vaidya, N.H., Pradhan, D.K.: A cluster-based approach for routing in ad-hoc networks. In: Proceedings of the USENIX Symposium on Location Independent and Mobile Computing, pp. 1–8 (1995)
27. Leskovec, J.: Stanford network analysis project (2012). http://snap.stanford.edu/data/
28. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. J. Comput. Syst. Sci. **20**(2), 219–230 (1980)
29. Lund, C., Yannakakis, M.: The approximation of maximum subgraph problems. In: Proceedings of the 20th International Colloquium on Automata, Languages and Programming, ICALP '93, pp. 40–51. Springer, London (1993)
30. McClosky, B.: Independence systems and stable set relaxations. PhD thesis, Rice University (2008)
31. McClosky, B., Hicks, I.V.: The co-2-plex polytope and integral systems. SIAM J. Discrete Math. **23**(3), 1135–1148 (2009)
32. McClosky, B., Hicks, I.V.: Combinatorial algorithms for the maximum k-plex problem. J. Comb. Optim. **23**, 29–49 (2012)
33. Moser, H., Niedermeier, R., Sorge, M.: Exact combinatorial algorithms and experiments for finding maximum k-plexes. J. Combin. Optim., 1–27 (2011). doi:10.1007/s10878-011-9391-5
34. Östergård, P.R.J.: A new algorithm for the maximum-weight clique problem. Electron. Notes Discrete Math. **3**, 153–156 (1999)
35. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Appl. Math. **120**, 197–207 (2002)
36. Östergård, P.R.J., Vaskelainen, V.P.: Russian Doll search for the Steiner triple covering problem. Optim. Lett. **5**(4), 631–638 (2011)
37. Pattillo, J., Youssef, N., Butenko, S.: On clique relaxation models in network analysis. Eur. J. Oper. Res. **226**, 9–18 (2013)
38. Ramaswami, R., Parhi, K.K.: Distributed scheduling of broadcasts in a radio network. In: Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '89), vol. 2, pp. 497–504 (1989)
39. Scott, J.: Social Network Analysis: A Handbook, 2nd edn. Sage Publications, London (2000)
40. Seidman, S.B., Foster, B.L.: A graph theoretic generalization of the clique concept. J. Math. Sociol. **6**, 139–154 (1978)
41. Sewell, E.C.: A branch and bound algorithm for the stability number of a sparse graph. INFORMS J. Comput. **10**(4), 438–447 (1998)
42. Sloane, N.J.A.: Unsolved problems in graph theory arising from the study of codes. Graph Theory Notes N. Y. **18**, 11–20 (1989)
43. Sloane, N.J.A.: Challenge problems: Independent sets in graphs (2000). Online: http://www.research.att.com/~njas/doc/graphs.html. Accessed July 2003
44. Sloane, N.J.A.: On single-deletion-correcting codes. In: Arasu, K.T., Seress, A. (eds.) Codes and Designs. Ohio State University Mathematical Research Institute Publications, vol. 10, pp. 273–291. Walter de Gruyter, Berlin (2002)
45. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **37**(1), 95–111 (2007)
46. Vaskelainen, V.: Russian Doll Search algorithms for discrete optimization problems. PhD thesis, Helsinki University of Technology (2010)
47. Verfaillie, G., Lemaitre, M., Schiex, T.: Russian Doll Search for solving constraint optimization problems. In: Proceedings of the National Conference on Artificial Intelligence, pp. 181–187. Citeseer, Princeton (1996)
48. Wasserman, S., Faust, K.: Social Network Analysis. Cambridge University Press, New York (1994)
49. Wood, D.R.: An algorithm for finding a maximum clique in a graph. Oper. Res. Lett. **21**(5), 211–217 (1997)
50. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: STOC '78: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 253–264. ACM Press, New York (1978)
51. Yannakakis, M.: The effect of a connectivity requirement on the complexity of maximum subgraph problems. J. ACM **26**(4), 618–630 (1979)
52. Yu, H., Paccanaro, A., Trifonov, V., Gerstein, M.: Predicting interactions in protein networks by completing defective cliques. Bioinformatics **22**(7), 823–829 (2006)