Decremental Clique Problem [⋆]

Fabrizio Grandoni and Giuseppe F. Italiano

Dipartimento di Informatica, Sistemi e Produzione
Università di Roma “Tor Vergata”
Via del Politecnico 1
00133 Roma, Italy
{grandoni,italiano}@disp.uniroma2.it

Abstract. The clique problem consists in determining whether an undirected graph G of order n contains a clique of order ℓ. In this paper we are concerned with the decremental version of clique problem, where the property of containing an ℓ-clique is dynamically checked during deletions of nodes. We provide an improved dynamic algorithm for this problem for every fixed value of $\ell \geq 3$. Our algorithm naturally applies to filtering for the constraint satisfaction problem. In particular, we show how to speed up the filtering based on an important local consistency property: the inverse consistency.

1 Introduction

There is a wealth of research on *dynamic graph problems*, which consist in checking a given property on graphs subject to dynamic changes, such as deletions or insertions of nodes or edges [4,5,6,9,10,13,14,18,19]. If only deletions or insertions are allowed, the dynamic problem is also called *decremental* or *incremental* respectively.

A *clique* is an undirected graph such that its nodes are pairwise adjacent. The *decremental clique problem* consists in dynamically determining whether a graph G of n nodes contains an ℓ-clique (a clique of ℓ nodes), during deletions of nodes.

To the best of our knowledge, no non-trivial algorithm is known for this problem, while several non-trivial results are available for its static version. Itai and Rodeh [12] showed how to detect a *triangle* (clique of three nodes) in G in $O(n^{\omega})$ steps, where the complexity of multiplying two $n \times n$ matrices is $\mathrm{O}(n^{\omega})$, $\omega < 2.376$ [1]. Nešetřil and Poljak [17] generalized the algorithm of Itai and Rodeh to the detection of cliques of arbitrary cardinality ℓ. Their algorithm has a $O(n^{\alpha(\ell)})$ time complexity, where $\alpha(\ell) = \omega\lfloor \ell/3 \rfloor + \ell \pmod 3$.

Recently, Eisenbrand and Grandoni [7] developed a faster algorithm for the same task. Their algorithm has a $\mathrm{O}(n^{\beta(\ell)})$ time complexity, where $\beta(\ell) = \omega(\lfloor \ell/3 \rfloor, \lceil (\ell-1)/3 \rceil, \lceil \ell/3 \rceil)$, and the time complexity of multiplying a $n^r \times n^s$ matrix by a $n^s \times n^t$ matrix is denoted by $\mathrm{O}(n^{\omega(r,s,t)})$.

All the algorithms above can be easily adapted so as to count the number of ℓ-cliques in which each node is contained.

In this paper we present a dynamic algorithm for the decremental clique problem. In particular, we show how to efficiently update the number of ℓ-cliques in which each node of a graph is contained, during deletions of nodes. Our algorithm, which builds up on the algorithm of Eisenbrand and Grandoni, performs updates in $\mathrm{O}(n^{\beta(\ell)-0.8})$ time for every fixed ℓ, that is, roughly, $n^{0.8}$ times faster than recomputing everything from scratch.

1.1 An Application to the Constraint Satisfaction Problem

The *constraint satisfaction problem* consists in determining whether a set of k variables, defined on domains of size at most d, admits an instantiation which satisfies a given set of constraints.

[⋆] This work has been partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT), by the Italian Ministry of University and Research (Project “ALINWEB: Algorithmics for Internet and the Web”).

Any such instantiation is a *solution* for the *constraint network*. Without loss of generality [16], we can assume that all the constraints are *binary* (a constraint is binary if it involves only a pair of variables).

An assignment (i, a) of a value a to a variable i is *consistent* if there is a solution which assigns a to i, and *inconsistent* otherwise. Inconsistent assignments can be removed from the constraint network without loosing any solution (by removing an assignment (i, a), we mean removing a from the domain of i).

Detecting inconsistent assignments is a NP-hard problem [16]. For this reason, many heuristic filtering techniques have been developed, which allow to efficiently detect (and remove) part of the inconsistent assignments. Most of them are based on some kind of *local consistency property* $\mathcal{P}$, which all the consistent assignments *need* to satisfy. The assignments which do not satisfy $\mathcal{P}$ are iteratively filtered out.

Note that an assignment which initially satisfies $\mathcal{P}$, may not satisfy $\mathcal{P}$ any more after some deletions. Thus the same assignment may be checked for consistency many times along the filtering process. This suggests the idea of performing such repeated checks dynamically, instead of doing it each time from scratch. In fact, this approach is used by most of the fastest filtering algorithms.

Maybe the simplest and most studied local consistency property is *arc consistency* [15]. An assignment (i, a) is arc consistent if, for every other variable j, there is at least one assignment (j, b) compatible with (i, a). Clearly, if a node is not arc consistent, it cannot be consistent (unless i is the unique variable in the network).

Arc consistency can be easily generalized. An assignment (i, a) is *path-inverse consistent* [8] if, for every other two variables j and h, there are assignments (j, b) and (h, c) which are mutually compatible and compatible with (i, a). The ℓ-inverse consistency [8] is the natural generalization of arc-consistency ($\ell = 2$) and path-inverse consistency ($\ell = 3$) to arbitrary (fixed) values of $\ell \leq k$.

The currently fastest filtering algorithm based on ℓ-inverse consistency is the $\mathrm{O}(k^\ell d^\ell)$ algorithm of Debruyne [3]. This algorithm is based on a very simple dynamic strategy to check whether an assignment is ℓ-inverse consistent.

We show how to reduce the problem of dynamically checking ℓ-inverse consistency to the decremental clique problem on graphs of $\mathrm{O}(d)$ nodes. By applying this reduction and our $\mathrm{O}(d^{\beta(\ell)-0.8})$ decremental algorithm, we reduce the complexity of ℓ-inverse consistency based filtering to $\mathrm{O}(k^\ell d^{\beta(\ell)+0.2})$. This implies an improvement for every $\ell \geq 3$.

The remainder of this paper is organized as follows. In Section 2 we introduce some preliminaries. In Section 3 we describe the algorithm of Eisenbrand and Grandoni, upon which our decremental algorithm builds up. In Section 4 we present our decremental algorithm to maintain the number of ℓ-cliques. Eventually, in Section 5 we show how to speed up the filtering based on inverse consistency.

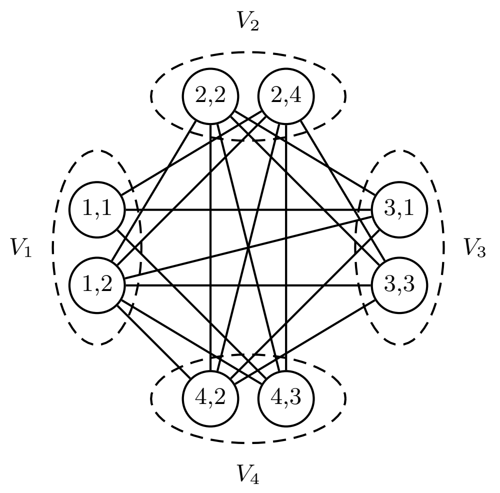
2 Preliminaries

We use standard graph notation as contained for instance in [2]. An *undirected graph* G is a pair (V, E), where V is a finite set of *nodes* and the *edge* set E consists of unordered pairs of nodes. Without loss of generality, we can assume $V = \{1, 2 \dots |V|\}$, where $|V|$ is the *cardinality* of G. Two nodes v and w are *adjacent* if $\{v, w\} \in E$. A graph is *complete* if each pair of distinct nodes is adjacent. An *ℓ-clique* is a complete graph of ℓ nodes. The 3-cliques are also called *triangles*. The graph $G[V']$ *induced on* G by a subset V' of nodes is the graph obtained from G by removing all the nodes not in V' and the edges incident on them.

The adjacency matrix A of G is a 0-1 matrix such that, for each pair of nodes v and w, $A[v, w] = 1$ if and only if v and w are adjacent (in particular A is symmetric and the main diagonal is set to zero).

A k-partite graph $G = (\{V_1, V_2 \dots V_k\}, E)$ is a graph where the set of nodes is $V = \bigcup_i V_i$, the set of edges is E, the subsets V_i (*partitions*) are disjoint, and the nodes in the same partition are not adjacent.

Figure 1 Example of consistency graph. The partitions corresponding to each variable are included into dashed ellipses. A solution is given by the assignments $(1,2)$, $(2,2)$, $(3,1)$ and $(4,2)$. The assignment $(1,1)$ is arc consistent, while it is not path-inverse consistent.



A binary constraint network of k variables can be naturally represented via a k-partite graph $G = (\{V_1, V_2 \dots V_k\}, E)$, the *consistency graph*, which has a node for each possible assignment (i, a) and an edge between all the pairs of assignments which are compatible according to the constraints. In particular, partition V_i is formed by all the assignments corresponding to variable i (two values cannot be assigned to the same variable). An example of consistency graph is given in Figure 1.

It is not hard to show that a k-clique in G corresponds to each solution of the binary constraint network. In other words, the binary constraint satisfaction problem is equivalent to the problem of determining whether the consistency graph contains a k-clique.

The definitions concerning the assignments can be naturally extended to the nodes of the consistency graph. In particular, a node (i, a) is *consistent* if it belongs to at least one k-clique, and *inconsistent* otherwise. Node (i, a) is *ℓ-inverse consistent*, $\ell \leq k$, if, taken ℓ partitions $V_{j_1}, V_{j_2} \dots V_{j_\ell}$ including V_i, node (i, a) is contained in at least one ℓ-clique of the graph $G[\cup_k V_{j_k}]$ induced on G by such partitions. Clearly, if a node is not ℓ-inverse consistent, it cannot be consistent.

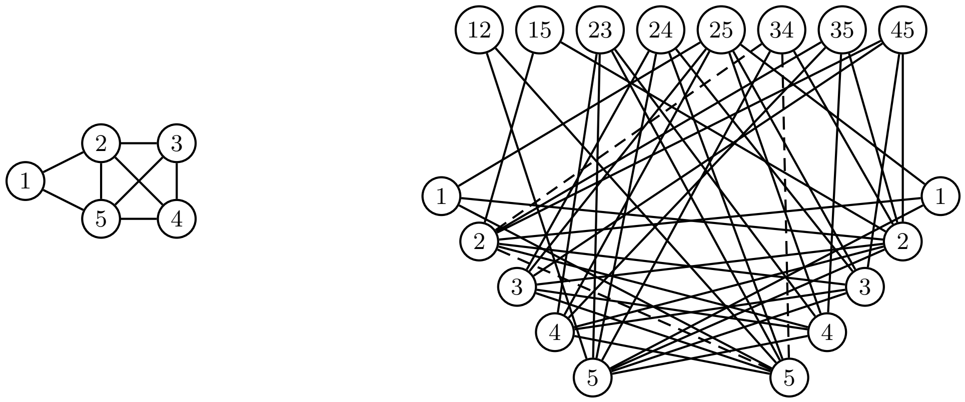
3 Static Algorithm

In this section we describe a static algorithm to count the number of ℓ-cliques in which each node of an undirected graph $G = (V, E)$, with n nodes, is contained.

We adapt the algorithm of Eisenbrand and Grandoni for the clique problem to this purpose. We first recall their algorithm. They compute the following 3-partite auxiliary graph $\widetilde{G}_\ell = (\{W_1, W_2, W_3\}, F) = \widetilde{G}$. Let ℓ_1, ℓ_2 and ℓ_3 be equal to $\lfloor \ell/3 \rfloor$, $\lceil (\ell-1)/3 \rceil$ and $\lceil \ell/3 \rceil$ respectively (notice that $\ell = \ell_1 + \ell_2 + \ell_3$). Partition W_i, $i \in \{1, 2, 3\}$, is formed by the ℓ_i-cliques of G. A node $w_i \in W_i$ is adjacent to a node $w_j \in W_j$, if $i \neq j$ and the nodes of w_i and w_j induce a clique of order $(\ell_i + \ell_j)$ in G. Then the algorithm return *yes* if and only if $\widetilde{G}$ contains a triangle. In Figure 2 an example of graph with the corresponding auxiliary graph in the case $\ell = 4$ is depicted.

Lemma 1. *For every fixed $\ell \geq 3$, the algorithm above determines whether an undirected graph G of n nodes contains a clique of ℓ nodes in time* $O(n^{\beta(\ell)}) = O(n^{\omega(\lfloor \ell/3 \rfloor, \lceil (\ell-1)/3 \rceil, \lceil \ell/3 \rceil)})$ *time.*

Figure 2 An example of graph G (on the left) with the corresponding auxiliary graph $\widetilde{G}$ in the case $\ell = 4$. The nodes of $\widetilde{G}$ are labelled with the corresponding subset of nodes of G. One of the $\binom{4}{1,1,2} = 12$ triangles of $\widetilde{G}$ corresponding to the clique $\{2, 3, 4, 5\}$ of G is pointed out via dashed lines.



Proof. Let $\widetilde{G}$ be the auxiliary graph defined above. We show that G contains a ℓ-clique if and only if $\widetilde{G}$ contains a triangle. Assume that G contains a ℓ-clique $\{v_1, v_2 \dots v_\ell\}$. Thus the partitions W_1, W_2 and W_3 of $\widetilde{G}$ contain the nodes $w_1 = \{v_1, v_2 \dots v_{\ell_1}\}$, $w_2 = \{v_{\ell_1+1}, v_{\ell_1+2} \dots v_{\ell_1+\ell_2}\}$ and $w_3 = \{v_{\ell_1+\ell_2+1}, v_{\ell_1+\ell_2+2} \dots v_\ell\}$ respectively. Moreover w_1, w_2 and w_3 are pairwise adjacent. Thus $\widetilde{G}$ contains a triangle.

Assume now that $\widetilde{G}$ contains a triangle $\{w_1, w_2, w_3\}$. Let $T = \bigcup_i w_i$. Since the graph is 3-partite, the nodes w_i must belong to distinct partitions. Moreover two nodes of $\widetilde{G}$ which contain the same node v of G cannot be adjacent. Thus $|T| = \ell_1 + \ell_2 + \ell_3 = \ell$. Every two distinct nodes of T are adjacent in G. Thus T is a subset of ℓ pairwise adjacent nodes of G, that is a ℓ-clique of G.

Consider now the time complexity of the algorithm. Partition W_i contains $\mathrm{O}(n^{\ell_i})$ nodes, $i \in \{1, 2, 3\}$. A triangle of $\widetilde{G}$ can be detected in the following way. For each pair of nodes $\{w_1, w_3\}$, $w_1 \in W_1$ and $w_3 \in W_3$, one computes the number $P_{1,2,3}(w_1, w_3)$ of 2-length paths of the kind (w_1, w_2, w_3), where $w_2 \in W_2$. The value of $P_{1,2,3}$ is obtained by multiplying the $n^{\ell_1} \times n^{\ell_2}$ adjacency matrix of the nodes in W_1 with the nodes in W_2 by the $n^{\ell_2} \times n^{\ell_3}$ adjacency matrix of the nodes in W_2 with the nodes in W_3. Graph $\widetilde{G}$ contains a triangle if and only if there is a pair of adjacent nodes $\{w_1, w_3\}$, $w_1 \in W_1$ and $w_3 \in W_3$, such that $P_{1,2,3}(w_1, w_3) > 0$. Computing $P_{1,2,3}$ costs $= \mathrm{O}(n^{\omega(\ell_1,\ell_2,\ell_3)}) = \mathrm{O}(n^{\beta(\ell)})$. This is also an upper bound on the complexity of the algorithm.

A rectangular matrix multiplication can be executed through a straightforward decomposition into square blocks and fast square matrix multiplication. In other words:

$$\omega(r, s, t) \leq r + s + t + (\omega - 3)\min\{r, s, t\}.$$

More sophisticated fast rectangular matrix multiplication algorithms are available. In particular, for every $0 \leq r \leq 1$, the following bound holds [1,11]:

$$\omega(1, 1, r) \leq \begin{cases} 2 + o(1) & \text{if } 0 \leq r \leq \alpha = 0.294; \\ \omega - (1 - r)\frac{\omega - 2}{1 - \alpha} & \text{if } \alpha < r \leq 1. \end{cases} \tag{1}$$

With this bound at hand, one obtains:

$$\beta(\ell) \le \begin{cases} \lfloor\frac{\ell}{3}\rfloor\omega & \text{if } \ell \pmod 3 = 0; \\ \lfloor\frac{\ell}{3}\rfloor\omega + 1 & \text{if } \ell \pmod 3 = 1; \\ \lfloor\frac{\ell}{3}\rfloor\omega + 2 - \frac{\alpha(\omega-2)}{1-\alpha} & \text{if } \ell \pmod 3 = 2. \end{cases}$$

Better bounds are available for $\omega(r,s,t)$ [11]. These bounds lead to a tighter bound on $\beta(\ell)$ in the case $\ell \pmod 3 \neq 0$. For simplicity we will not consider these tighter bounds, since they are not expressed via a closed formula.

3.1 Counting Cliques

Consider now the problem of counting the number $K_\ell(v)$ of ℓ-cliques in which each node v of G is contained. The algorithm of Eisenbrand and Grandoni can be easily adapted to count, in $\mathrm{O}(n^{\beta(\ell)})$ time, the number $\widetilde{K}_3(w)$ of triangles in which each node w of $\widetilde{G}$ is contained. Note that many triangles in $\widetilde{G}$ may correspond to the same ℓ-clique of G.

More precisely, the number of distinct triangles of $\widetilde{G}$ which correspond to the same ℓ-clique of G is equal to the number of ways in which one can partition a set of cardinality ℓ in three subsets of cardinality ℓ_1, ℓ_2 and ℓ_3 respectively, that is $\binom{\ell}{\ell_1,\ell_2,\ell_3}$.

Let $W_i(v)$, for each $i \in \{1,2,3\}$, be the set of nodes of W_i which contain node v. It is not hard to show that the sum of $\widetilde{K}_3(w)$ over $W_1(v)$ is equal to $K_\ell(v)$, multiplied by the number of ways in which one can partition a set of cardinality $(\ell-1)$ in three subsets of cardinality (ℓ_1-1), ℓ_2 and ℓ_3 respectively:

$$\sum_{w\in W_1(v)} \widetilde{K}_3(w) = \binom{\ell-1}{\ell_1-1,\ell_2,\ell_3} K_\ell(v). \tag{2}$$

Then we can compute $K_\ell(v)$, for each $v \in V$, in $\mathrm{O}(n^{\beta(\ell)})$ steps.

Corollary 1. *The algorithm above counts the number of cliques of ℓ nodes in which each node of an undirected graph G of n nodes is contained, in time $\mathrm{O}(n^{\beta(\ell)})$.*

4 Decremental Algorithm

In this section we consider the problem of decrementally updating the number $K_\ell(v)$ of cliques of cardinality ℓ in which each node v of the undirected graph G is contained, during deletions of nodes.

The idea is to update the value of $\widetilde{K}_3(w)$, for each w in W_1, and then update $K_\ell(v)$, for each v in w, following Equation (2). Consider the deletion of a node u. The deletion of u corresponds to the deletion of the subsets of nodes $W_1(u)$, $W_2(u)$ and $W_3(u)$ in W_1, W_2 and W_3 respectively.

As two nodes of $\widetilde{G}$ which contain the same node u of G cannot belong to the same triangle, one can safely consider the effects of the deletion of each node in $W_i(u)$, $i \in \{1,2,3\}$, separately. First of all, for each $w \in W_1(u)$, one sets $\widetilde{K}_3(w)$ to zero (in linear time).

Consider now the deletion of the nodes in $W_2(u)$ (the deletion of the nodes in $W_3(u)$ is completely analogous). For each $w_1 \in W_1$ and for each deleted node $w_2 \in W_2(u)$, the value of $\widetilde{K}_3(w_1)$ has to be decreased by the number of triangles in which both nodes w_1 and w_2 are contained (at the same time). This quantity is zero if w_1 and w_2 are not adjacent, and it is equal to the number $P_{1,3,2}(w_1,w_2)$ of 2-length paths from w_1 to w_2 through a node in W_3 otherwise. Then one needs to compute $P_{1,3,2}(w_1,w_2)$, for each $w_1 \in W_1$ and for each $w_2 \in W_2(u)$. A simple-minded approach is to compute the number of such paths from scratch.

A better time bound can be obtained as follows. One maintains a *lazy* value $P'_{1,3,2}(w_1,w_2)$ of $P_{1,3,2}(w_1,w_2)$, for each $w_1 \in W_1$ and $w_2 \in W_2$. Whenever a node w_3 is removed from W_3, instead of updating $P'_{1,3,2}$, one stores w_3 in a set D_3. When the cardinality of D_3 reaches a given threshold, one updates $P'_{1,3,2}$ and empties D_3. Clearly the current value of $P_{1,3,2}(w_1, w_2)$ depends on both $P'_{1,3,2}(w_1, w_2)$ and D_3.

In more details, one initially sets $D_3 = \emptyset$ and $P'_{1,3,2} = P_{1,3,2}$. Let $\mu_3 \in [0,1]$ be a parameter to be fixed later. When one removes a node w_3 from W_3, w_3 is added to D_3 and, if $|D_3| > n^{\ell_3 - 1 + \mu_3}$, one executes the following steps:

– $P'_{1,3,2}$ is updated by subtracting from $P'_{1,3,2}(w_1, w_2)$ the number $\Delta P_{1,3,2}(w_1, w_2)$ of 2-length paths from w_1 to w_2 through a node in D_3.
– Set D_3 is emptied.

The current value of $P_{1,3,2}(w_1, w_2)$, for every $w_1 \in W_1$ and $w_2 \in W_2$, is given by:

$$P_{1,3,2}(w_1, w_2) = P'_{1,3,2}(w_1, w_2) - \Delta P_{1,3,2}(w_1, w_2). \tag{3}$$

Let

$$\begin{aligned}\widetilde{\beta}(\ell) = \min_{\mu_2, \mu_3 \in [0,1]} \max\{&\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2) - \mu_3, \omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2 - 1), \\ &\omega(\ell_1, \ell_2 - 1 + \mu_2, \ell_3) - \mu_2, \omega(\ell_1, \ell_2 - 1 + \mu_2, \ell_3 - 1)\}.\end{aligned} \tag{4}$$

Theorem 1. *The algorithm above maintains the number of cliques of fixed cardinality ℓ in which each node of a graph of n nodes is contained, during deletion of nodes. The preprocessing time of the algorithm is $\mathrm{O}(n^{\beta(\ell)})$ and its amortized update time per deletion is $\mathrm{O}(n^{\widetilde{\beta}(\ell)})$.*

Proof. The correctness of the algorithm is a consequence of Equations (2) and (3).

Consider now the time complexity of the algorithm. Set $W_i(u)$, $i \in \{1,2,3\}$, contains $\mathrm{O}(n^{\ell_i - 1})$ nodes. The number $\Delta P_{1,3,2}(w_1, w_2)$ of 2-length paths from w_1 to w_2 through a node in D_3 can be obtained by multiplying the $n^{\ell_1} \times n^{\ell_3 - 1 + \mu_3}$ adjacency matrix of the nodes in W_1 with the nodes in D_3 by the $n^{\ell_3 - 1 + \mu_3} \times n^{\ell_2}$ adjacency matrix of the nodes in D_3 with the nodes in W_2. This costs $\mathrm{O}(n^{\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2)})$. The value of $P'_{1,3,2}$ can be updated within the same time bound. Note that the deletion of $\mathrm{O}(n^{\ell_3 - 1})$ nodes in W_3 corresponds to each deletion of one node in G. This means that one updates $P'_{1,3,2}$ every $\Omega(n^{\mu_3})$ deletions in G. Then the amortized update cost per deletion is $\mathrm{O}(n^{\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2) - \mu_3})$.

Following Equation (3), the current value of $P_{1,3,2}(w_1, w_2)$, for every $w_1 \in W_1$ and $w_2 \in W_2(u)$, can be computed in $\mathrm{O}(n^{\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2 - 1)})$ time. This is the time required to multiply the $n^{\ell_1} \times n^{\ell_3 - 1 + \mu_3}$ adjacency matrix of the nodes in W_1 with the nodes in D_3 by the $n^{\ell_3 - 1 + \mu_3} \times n^{\ell_2 - 1}$ adjacency matrix of the nodes in D_3 with the nodes in $W_2(u)$.

Then the cost of updating $\widetilde{K}_3(w)$, for each $w \in W_1$, after the deletion of the nodes in $W_2(u)$, is $\mathrm{O}(n^{\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2) - \mu_3} + n^{\omega(\ell_1, \ell_3 - 1 + \mu_3, \ell_2 - 1)})$. Analogously, the cost of updating $\widetilde{K}_3(w)$ after the deletion of the nodes in $W_3(u)$ is $\mathrm{O}(n^{\omega(\ell_1, \ell_2 - 1 + \mu_2, \ell_3) - \mu_2} + n^{\omega(\ell_1, \ell_2 - 1 + \mu_2, \ell_3 - 1)})$, where $\mu_2 \in [0,1]$. The claim follows by fixing μ_2 and μ_3 as suggested by Equation (4).

In next section we will show how to choose μ_2 and μ_3 so as to minimize the complexity of the algorithm.

4.1 Bounds on the Complexity

In this subsection we want to estimate the complexity of the decremental algorithm described above according to the current best bounds on rectangular matrix multiplication. For this purpose, we will use the bounds on $\omega(r,s,t)$ given by Equation (1) and by the following equation [11]. For any $0 \le t \le 1 \le r$:

$$\omega(t,1,r) \le \begin{cases} r + 1 + o(1) & 0 \le t \le \alpha; \\ r + 1 + (t - \alpha)\frac{\omega - 2}{1 - \alpha} + o(1) & \alpha < t \le 1. \end{cases} \tag{5}$$

With simple calculations involving Equations (1) and (5) one obtains the results of Table 1 and 2.

ℓ	μ_2 and μ_3
3	$\mu_2 = \mu_3 = \frac{1+\alpha-\alpha\omega}{4-2\alpha-\omega}$
4	$\mu_2 = \frac{1-\alpha(3-\omega)}{\omega-1-\alpha}$; $\mu_3 = \omega - 2$
$3k+2 \geq 5$	$\mu_2 = \mu_3 = \frac{1-\alpha}{1+\alpha(\omega-3)}$
$3k \geq 6$	$\mu_2 = \mu_3 = \frac{1+\alpha-\alpha\omega}{4-\alpha\omega-\omega}$
$3k+1 \geq 7$	$\mu_2 = 1$; $\mu_3 = \alpha\frac{\omega-2}{1-\alpha}$

Table 1: Optimal values of μ_2 and μ_3 for varying ℓ.

ℓ	$\widetilde{\beta}(\ell)$
3	$\frac{5-\alpha-\omega(\alpha+1)}{4-2\alpha-\omega}$
$3k+1 \geq 4$	$k\omega$
$3k+2 \geq 5$	$k\omega + \frac{1-\alpha}{1+\alpha(\omega-3)}$
$3k \geq 6$	$k\omega - \omega + 1 + (\frac{1+\alpha-\alpha\omega}{4-\alpha\omega-\omega})(1 - \alpha\frac{\omega-2}{1-\alpha})$

Table 2: Values of $\widetilde{\beta}(\ell)$ for varying ℓ.

Summarizing these results, the update cost is $O(n^{\widetilde{\beta}(\ell)}) = O(n^{\beta(\ell)-\delta(\ell)})$, where:

$$\delta(\ell) \geq \begin{cases} 0.800 & \text{if } \ell = 3; \\ 0.832 & \text{if } \ell \pmod 3 = 0,\ \ell \geq 6; \\ 1.000 & \text{if } \ell \pmod 3 = 1; \\ 0.978 & \text{if } \ell \pmod 3 = 2. \end{cases}$$

Thus our algorithm performs updates roughly $n^{0.8}$ times faster than recomputing everything from scratch.

In Table 3, the complexity of the decremental algorithm is compared with the complexity of the static algorithm for $3 \leq \ell \leq 8$.

ℓ	Static [7]	Dynamic
3	$O(n^{2.376})$	$O(n^{1.575})$
4	$O(n^{3.376})$	$O(n^{2.376})$
5	$O(n^{4.220})$	$O(n^{3.241})$
6	$O(n^{4.751})$	$O(n^{3.919})$
7	$O(n^{5.751})$	$O(n^{4.751})$
8	$O(n^{6.595})$	$O(n^{5.616})$

Table 3: Running time comparison of the static and dynamic algorithms for counting the cliques of cardinality ℓ.

5 Fast Inverse Consistency

Let G be a consistency graph with k partitions of size at most d. Consider the problem of computing the largest (for number of nodes) induced subgraph $G_{\ell\text{-IC}}$ of G such that all its partitions are non-empty and all its nodes are ℓ-inverse consistent, or determine that such graph does not exist. This problem is well defined:

Lemma 2. *Let $\mathcal{G}_{\ell\text{-}IC}$ be the set of all the induced subgraphs of G such that all their partitions are non-empty and all their nodes are ℓ-inverse consistent. If $\mathcal{G}_{\ell\text{-}IC}$ is not empty, it contains a unique graph $G_{\ell\text{-}IC}$ of maximum cardinality.*

Proof. Suppose that there exist two distinct graphs $G_1 = G[V_1]$ and $G_2 = G[V_2]$ in $\mathcal{G}_{\ell\text{-IC}}$ of maximum cardinality. Then $G' = G[V_1 \cup V_2]$ is an induced subgraph of G, of cardinality strictly greater than G_1 and G_2, whose nodes are ℓ-inverse consistent. This is a contradiction.

The fastest algorithm known to solve this problem [3] has a $O(k^\ell d^\ell)$ time complexity. In this section we present a faster algorithm for the same problem, which is based on the decremental algorithm of Section 4. Its time complexity is $\mathrm{O}(k^\ell d^{\widetilde{\beta}(\ell)+1}) = \mathrm{O}(k^\ell d^{\beta(\ell)+0.2})$. This improves on the $\mathrm{O}(k^\ell d^\ell)$ bound for any $\ell \geq 3$.

Our algorithm works as follows. Nodes which are not ℓ-inverse consistent are removed from G one by one. The procedure ends when all the nodes in G are ℓ-inverse consistent or a partition becomes empty. In the first case, at the end of the procedure G is equal to $G_{\ell\text{-IC}}$. In the second case, $\mathcal{G}_{\ell\text{-IC}}$ is empty.

We have to show how nodes which are not ℓ-inverse consistent are detected along the way. First of all, one checks all the nodes and removes the nodes which are not ℓ-inverse consistent. Then one has to propagate efficiently the effects of deletions. In fact, the deletion of one node can induce as a side effect the deletion of other nodes (which were previously recognized as ℓ-inverse consistent).

Consider the deletion of a node $v \in V_i$. Let $\mathcal{G}_\ell(i)$ be the set of graphs induced on G by the nodes of ℓ distinct partitions, including partition V_i. For each graph G' in $\mathcal{G}_\ell(i)$ and for each node w of G', one has to check whether w belongs to at least one ℓ-clique of G'.

In more details, a set `DelSet` of integers is used to keep trace of the partitions into which a deletion occurred: whenever a node v in a partition V_i is removed, i is stored in `DelSet`. We can distinguish two main steps in the algorithm: an initialization step and a propagation step. In the initialization step, for each node v in each partition V_i, one checks for each G' in $\mathcal{G}_\ell(i)$ whether v is contained in at least one ℓ-clique of G'. If this is not true, v is removed from G. In the propagation step, until `DelSet` is not empty, one extracts an integer j from `DelSet` and executes the following steps. For each G' in $\mathcal{G}_\ell(j)$ and for each node v in G', one checks whether v is contained in at least one ℓ-clique of G'. If not, v is removed from G.

We have to show how to check whether a node of a graph G' is contained in at least one ℓ-clique. The idea is to use the algorithm of previous section. For each graph G' induced by ℓ distinct partitions, we maintain the number of ℓ-cliques in which each one of its nodes is contained. Whenever a node v in a partition V_i is removed, one updates consequently these quantities for each graph G' in $\mathcal{G}_\ell(i)$.

Theorem 2. *The algorithm above computes $G_{\ell\text{-}IC}$ or determines that it does not exist in time $\mathrm{O}(k^\ell d^{\widetilde{\beta}(\ell)+1})$.*

Proof. The number of iterations of the propagation step is bounded by the number of nodes. Then the algorithm halts.

An ℓ-inverse consistent node is clearly never removed. Consider the non-trivial case that the algorithm halts when no partition is empty. To show correctness, we prove that all the remaining nodes in G are ℓ-inverse consistent. Assume by contradiction that, when the algorithm halts, G contains a node $v \in V_i$ which is not ℓ-inverse consistent.

Since all the nodes which are not ℓ-inverse consistent in the original graph are removed during the initialization step, v must be not ℓ-inverse consistent because of the deletions which occurred during the initialization and/or the propagation step.

Consider the sequence $v^{(1)}, v^{(2)} \dots v^{(p)}$ in which nodes are removed from the graph. Let q, $q \in \{1, 2 \dots p\}$, be the smallest index such that v is not ℓ-inverse consistent in the graph $G[V^{(q)}]$, where $V^{(q)} = V \backslash \{v^{(1)}, v^{(2)} \dots v^{(q)}\}$. Let V_j be the partition of $v^{(q)}$. Notice that $v^{(q)}$ must belong to a graph G' which contains partition V_i, and thus node v.

After the deletion of node $v^{(q)}$, j is inserted in `DelSet`. In one of the following steps, j is extracted from `DelSet` and all the nodes in any graph G' of $\mathcal{G}_\ell(j)$ are checked. In particular, node v is checked. Since in that iteration the set of nodes still in G is a subset of $V^{(q)}$, the node v is recognized as a node which is not ℓ-inverse consistent and it is thus removed, which is a contradiction.

The time complexity of the algorithm is bounded by the cost of maintaining the number of ℓ-cliques in which each node of each graph G' is contained. The number of such graphs is $O(k^{\ell})$ (that is the number of ways one can select ℓ from k partitions), and each graph contains $O(d)$ nodes. Then the total initialization cost is $O(k^{\ell} d^{\beta(\ell)})$. Since each graph G' is interested by at most $O(d)$ deletions, the total update cost is $O(k^{\ell} d^{\widetilde{\beta}(\ell)+1})$. Thus the time complexity of the algorithm is $O(k^{\ell}(d^{\beta(\ell)} + d^{\widetilde{\beta}(\ell)+1})) = O(k^{\ell} d^{\widetilde{\beta}(\ell)+1})$

The performance of our algorithm and of the previous best are compared in Table 4 for $3 \leq \ell \leq 8$. In particular, our algorithm reduces the time complexity to *enforce* path-inverse consistency from $O(k^3 d^3)$ to $O(k^3 d^{2.575})$.

ℓ	Previous best [3]	This paper
3	$O(k^3 d^3)$	$O(k^3 d^{2.575})$
4	$O(k^4 d^4)$	$O(k^4 d^{3.376})$
5	$O(k^5 d^5)$	$O(k^5 d^{4.241})$
6	$O(k^6 d^6)$	$O(k^6 d^{4.919})$
7	$O(k^7 d^7)$	$O(k^7 d^{5.751})$
8	$O(k^8 d^8)$	$O(k^8 d^{6.616})$

Table 4: Time complexity comparison of our algorithm to enforce ℓ-inverse consistency and the previous best.

References

1. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill Book Company, 2 edition, 2001.
3. R. Debruyne. A property of path inverse consistency leading to an optimal PIC algorithm. In *European Conference on Artificial Intelligence*, pages 88–92, 2000.
4. C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In *IEEE Symposium on Foundations of Computer Science*, pages 381–389, 2000.
5. C. Demetrescu and G. F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *IEEE Symposium on Foundations of Computer Science*, pages 260–267, 2001.
6. C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. In *ACM Symposium on the Theory of Computing*, pages 159–166, 2003.
7. F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. 2003. To appear in “Theoretical Computer Science”.
8. C. D. Elfe and E. C. Freuder. Neighborhood inverse consistency preprocessing. In *National Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, volume 1, pages 202–208, 1996.
9. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic shortest paths and negative cycles detection on digraphs with arbitrary arc weights. In *European Symposium on Algorithms*, pages 320–331, 1998.
10. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the Association for Computing Machinery*, 48(4):723–760, 2001.
11. X. Huang and V. Pan. Fast rectangular matrix multiplication and applications. *Journal of Complexity*, 14(2):257–299, 1998.
12. A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
13. V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *IEEE Symposium on Foundations of Computer Science*, pages 81–91, 1999.
14. V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. In *ACM Symposium on the Theory of Computing*, pages 492–498, 1999.
15. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
16. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

17. J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
18. A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *IEEE Symposium on Foundations of Computer Science*, pages 605–615, 1999.
19. U. Zwick. All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 310–319, 1998.