Incrementally Maintaining the Number of l-Cliques

Fabrizio Grandoni

MPI–I–2002–1-002 June 2002

Author's Address

Fabrizio Grandoni
Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken, Germany
grandoni@mpi-sb.mpg.de

Abstract

The main contribution of this paper is an incremental algorithm to update the number of l-cliques, for $l \geq 3$, in which each node of a graph is contained, after the deletion of an arbitrary node. The initialization cost is $O(n^{\omega p+q})$, where n is the number of nodes, $p = \lfloor \frac{l}{3} \rfloor$, $q = l \pmod 3$, and $\omega = \omega(1,1,1)$ is the exponent of the multiplication of two $n \times n$ matrices. The amortized updating cost is $O(n^q T(n,p,\varepsilon))$ for any $\varepsilon \in [0,1]$, where $T(n,p,\varepsilon) = \min\{n^{p-1}(n^{p(1+\varepsilon)} + n^{p(\omega(1,\varepsilon,1)-\varepsilon)}), n^{p\omega(1,\frac{p-1}{p},1)}\}$ and $\omega(1,r,1)$ is the exponent of the multiplication of an $n \times n^r$ matrix by an $n^r \times n$ matrix. The current best bounds on $\omega(1,r,1)$ imply an $O(n^{2.376p+q})$ initialization cost, an $O(n^{2.575p+q-1})$ updating cost for $3 \leq l \leq 8$, and an $O(n^{2.376p+q-0.532})$ updating cost for $l \geq 9$. An interesting application to constraint programming is also considered.

1 Introduction

Detecting triangles in graphs is a very basic problem in algorithms. In (Itai & Rodeh 1978), Itai and Rodeh showed an $O(n^{\omega})$ algorithm to decide if a given undirected graph G contains a triangle, where n is the number of nodes and $\omega = \omega(1,1,1)$ is the exponent of the multiplication of two $n \times n$ matrices (at the moment the best bound on ω is $2 \leq \omega < 2.376$ (Coppersmith & Winograd 1990)). Faster algorithms exist for sparse graphs (Alon, Yuster & Zwick 1997, Chiba & Nishizeki 1985, Itai & Rodeh 1978). In (Nesetril & Poljak 1985), Nesetril and Poljak generalized the algorithm of Itai and Rodeh to detect cliques of arbitrary size. Their algorithm decides if G contains an l-clique, for $l \geq 3$, in $O(n^{\omega p+q})$ time, where $p = \lfloor \frac{l}{3} \rfloor$ and $q = l \pmod 3$.

We here consider a slightly different problem, that we will call the *l-cliques covering problem*: "do the l-cliques cover all the nodes?". A simple variant of the algorithm described in (Nesetril & Poljak 1985) allows to answer to the l-cliques covering problem within the same time bound. The main contribution of this paper is an algorithm to incrementally answer to the l-cliques covering problem, after the deletion of an arbitrary node. In particular, our algorithm incrementally maintains the number $k_l(v)$ of l-cliques in which each node v is contained (node v is not covered by an l-clique if and only if $k_l(v) = 0$). The initialization cost is $O(n^{\omega p+q})$ and the amortized updating cost is $O(n^q T(n,p,\varepsilon)$ for any $\varepsilon \in [0,1]$, where $T(n,p,\varepsilon) = \min\{n^{p-1}(n^{p(1+\varepsilon)} + n^{p(\omega(1,\varepsilon,1)-\varepsilon)}), n^{p\omega(1,\frac{p-1}{p},1)}\})$ and $\omega(1,r,1)$ is the exponent of the multiplication of a $n \times n^r$ matrix by a $n^r \times n$ matrix. The current best bounds on $\omega(1,r,1)$ imply an $O(n^{2.575p+q-1})$ updating cost for $3 \leq l \leq 8$ and an $O(n^{2.376p+q-0.532})$ updating cost for $l \geq 9$ (Coppersmith & Winograd 1990, Huang & Pan 1998).

The significance of our result is highlighted via an application in constraint programming. Constraint programming is a declarative programming paradigm which allows to naturally formulate computational problems and which leads to interesting algorithmic questions as stressed in (Mehlhorn 2000). One of the basic problems in constraint programming is the binary constraint satisfaction problem, which consists in deciding if an instantiation of a finite set of variables, defined on finite domains, exists which satisfies a finite set of binary constraints. This (NP-complete) problem is equivalent to the problem of deciding if an undirected k-partite graph G contains a k-clique. Many filtering techniques have been developed to reduce in practice the complexity of this problem. One of the most important filtering techniques consists in computing the largest induced subgraph, $ic_l(G)$, of G which satisfies a particular property: the *l inverse consistency*, $2 \leq l \leq k$ (Freuder & Elfe 1996, Mackworth 1977). The fastest algorithm known to compute $ic_l(G)$ has a $O(k^l d^l)$ time complexity, where d is the size of the largest partitions (Bessiere 1994, Debruyne 2000). A straightforward consequence of the incremental algorithm to answer to the l-clique covering problem, is a faster algorithm to compute $ic_l(G)$ for $l \geq 3$ (the $O(k^2d^2)$ algorithm described in (Bessiere 1994) to compute $ic_2(G)$ is optimal). Its time complexity is $O(k^l d^{q+1} T(d,p,\varepsilon)$ for any $\varepsilon \in [0,1]$. The current best bounds on $\omega(1,r,1)$ imply an $O(k^l d^{2.575p+q})$ time complexity for $3 \leq l \leq 8$ and an $O(k^l d^{2.376p+q+0.468})$ time complexity for $l \geq 9$. This improves on the $O(k^l d^l)$ bound for any $l \geq 3$.

In Paragraph 2 we introduce some notation. In Paragraph 3 we consider the triangles covering problem and in Paragraph 4 its generalization to l-cliques, $l \geq 3$. Paragraph 5 is devoted to the new and faster l inverse consistency based filtering algorithm.

2 Preliminaries

We will use standard graph notation. Given a graph G, $V(G)$ is its set of nodes and $E(G)$ its set of edges. For any $v \in V(G)$, $N(v)$ is the neighborhood of v. The adjacency matrix A of G, is an integer matrix such that, for all v' and v'' in $V(G)$:

$$A[v', v''] = \begin{cases} 1 & \text{if } (v', v'') \in E(G), \\ 0 & \text{otherwise.} \end{cases}$$

We will assume, for simplicity, $A[v, v] = 0$ for all $v \in V(G)$. The graph $G[V']$ induced on G by a set V', is the subgraph of G obtained removing all the nodes not contained in V' and the corresponding edges. Graph G is complete if and only if, for all v' and v'' in $V(G)$, $v' \neq v''$, $(v', v'') \in E(G)$. An l-clique of G, $l \geq 1$, is a subset L of $V(G)$, of cardinality l, such that $G[L]$ is complete. Let K_l be the set of l-cliques of G and $K_l(v)$ be the set of l-cliques containing node v:

$$K_l(v) = \{h \in K_l : v \in h\}.$$

Let moreover $k_l(v)$ be the cardinality of $K_l(v)$:

$$k_l(v) = |K_l(v)|.$$

Whenever a confusion may occur, we indicate the graph considered with an apex (for example, $N^F(v)$ is the neighborhood of node v in graph F). We moreover do not distinguish, for simplicity, a node from the corresponding index in any one of the matrices considered. All matrix operations are executed in the ring of integers.

3 Incrementally Updating the Number of Triangles

The fastest algorithm known to decide if a given undirected graph G contains a triangle has an $O(n^{\omega})$ time complexity (Itai & Rodeh 1978), where $n = |V(G)|$ and ω is the exponent of the multiplication of two $n \times n$ matrices (at the moment the best bound on ω is $2 \leq \omega < 2.376$ (Coppersmith & Winograd 1990)). Actually, the procedure described in (Itai & Rodeh 1978) can also be used to compute the number $k_3(v)$ of triangles in which each node v is contained (within the same time bound). In fact, it is not hard to show that, for all $v \in V(G)$:

$$k_3(v) = \frac{1}{2} A^3[v, v]. \tag{1}$$

Consider now the problem of incrementally recomputing the function k_3 after the deletion of an arbitrary node (and of all the edges incident to it). The trivial approach is maintaining an integer matrix P such that $P[v', v'']$ contains the number of 2-length paths, in the current graph, between nodes v' and v''. The initial value of P is A^2 and we can update it in $O(n^2)$ time after each deletion. The number of triangles removed from $K_3(v)$, removing a node u, is equal to 0 if $(v, u) \notin E(G)$ and to $P[v, u]$ otherwise (where P is considered before the deletion of u). With this approach, the initialization cost is $O(n^{\omega})$ and the updating cost $O(n^2)$.

Is it possible to do better? The idea is not to update P after each deletion, but periodically and using fast rectangular matrix multiplication (a similar approach is used in other incremental graph algorithms (Demetrescu & Italiano 2000, Zwick 1998)). We use a data structure DS which consists of:

i. A set L, which contains deleted nodes,

ii. An $n \times n$ integer matrix D, which is equal to the current adjacency matrix not considering the deletions relative to the nodes in L,

iii. An $n \times n$ integer matrix P, which is equal to D^2,

iv. An n-dimensional vector C, such that $C[v] = k_3(v)$ for each node v.

In the initialization step, we set $L = \emptyset$, $D = A$, $P = A^2$, and $C[v] = k_3(v)$ for any $v \in V(G)$. This costs $O(n^{\omega})$. After the deletion of a node u, we set $C[u]$ to 0 and, for each node v still in $V(G)$, we decrease $C[v]$ of the number of triangles removed from $K_3(v)$ deleting u:

$$C[v] = C[v] - (P[v,u] - \sum_{w \in L} D[v,w] \cdot D[w,u]) \cdot D[v,u].$$

Then we add u to L and when $|L| \geq n^{\varepsilon}$, where ε is a real number in $[0,1]$, we execute the following operations:

i. We update $P[v',v'']$, for all v' and v'' in $V(G)$, $v' \neq v''$, according to the following rule:

$$P[v',v''] = P[v',v''] - \sum_{w \in L} D[v',w] \cdot D[w,v''].$$

This can be done in $O(n^{\omega(1,\varepsilon,1)})$ time, using fast rectangular matrix multiplication.

ii. We update D (in $O(n^{1+\varepsilon})$ time) setting to zero the rows and columns corresponding to the nodes in L.

iii. We empty L.

As updating C costs $O(n^{1+\varepsilon})$ and as we update D and P every $\Omega(n^{\varepsilon})$ deletions, the amortized updating cost is $O(n^{1+\varepsilon} + n^{\omega(1,\varepsilon,1)-\varepsilon})$.

Theorem 1. *There is an incremental procedure to update the number of triangles in which each node of a graph is contained, after the deletion of an arbitrary node, which has an $O(n^{\omega})$ initialization cost and an $O(n^{1+\varepsilon} + n^{\omega(1,\varepsilon,1)-1})$ updating cost for any $\varepsilon \in [0,1]$, where n is the number of nodes in the original graph.*

This time complexity is minimized when ε satisfies the following equation:

$$1 + 2\varepsilon = \omega(1,\varepsilon,1). \qquad (2)$$

We can use the following bound on $\omega(1,r,1)$ to fix ε (Huang & Pan 1998):

Theorem 2 (Huang and Pan 1998). *For all $r \in [0,1]$:*

$$\omega(1,r,1) \le \begin{cases} 2+o(1) & \text{if } 0 \le r \le \alpha, \\ r\frac{\omega-2}{1-\alpha} + \frac{2-\alpha\omega}{1-\alpha} & \text{if } \alpha < r \le 1, \end{cases}$$

where $\alpha = 0.294$.

From Theorem 2, we obtain $\varepsilon \le \frac{\alpha(\omega-1)-1}{\omega+2\alpha-4} < 0.575$, and then an $O(n^{1.575})$ amortized updating cost.

4 Incrementally Updating the Number of l-Cliques

We will now consider the generalization of the triangles covering problem to l-cliques, $l \ge 3$. In (Nesetril & Poljak 1985), an $O(n^{p\omega+q})$ procedure is described to decide if an undirected graph G contains an l-clique, $l \ge 3$, where $n = |V(G)|$, $p = \lfloor \frac{l}{3} \rfloor$ and $q = l \pmod 3$. The same basic idea can be used to compute the number $k_l(v)$ of l-cliques in which each node v is contained. Given a graph F, let $A^{F,k}$ be an auxiliary graph obtained:

i. Associating a node $v^{F,k}(h)$ to each k-clique h of F,

ii. Connecting a pair of nodes $v^{F,k}(h_1)$ and $v^{F,k}(h_2)$ if and only if the nodes of h_1 and h_2 form a $(2k)$-clique in F.

Notice that, if a pair of k-cliques in F share a node, the corresponding nodes in $A^{F,k}$ are not connected. The interesting property of $A^{F,k}$ is that F contains a $(3k)$-clique if and only if $A^{F,k}$ contains a triangle. In particular, to each $(3k)$-clique of F correspond a set of triangles whose cardinality is equal to the number of ways in which we can partition a set of $(3k)$ elements in 3 subsets of k elements each, that is:

$$\frac{1}{3!}\binom{3k}{k}\binom{2k}{k}\binom{k}{k} = \frac{(3k)!}{3!(k!)^3}.$$

Let $S^{F,k}(v)$ be the set of nodes in $A^{F,k}$ corresponding to the k-cliques of F which contain node $v \in V(F)$:

$$S^{F,k}(v) = \{v^{F,k}(h) \in V(A^{F,k}) : v \in h\}.$$

It is not hard to show that:

$$k_{3k}^{F}(v) = \frac{2(k!)^3}{(3k)!} \cdot \sum_{w \in S^{F,k}(v)} k_3^{A^{F,k}}(w). \tag{3}$$

This allows to reduce the problem of computing k_l^G to the problem of computing $k_3^{A^{G,\frac{l}{3}}}$ when $q = 0$. When $q \neq 0$, we can use the following simple recursive relation:

$$k_m^F(v) = \frac{1}{m-1} \sum_{w \in N^F(v)} k_{m-1}^{F[N^F(v)]}(w). \tag{4}$$

Equations 3 and 4 allow to reduce the problem of counting the number of l-cliques in the original graph to the problem of counting the number of triangles, using Equation 1, in $O(n^q)$ auxiliary graphs, each containing $O(n^p)$ nodes. The function k_l can then be computed in $O(n^{\omega p+q})$ time.

Now consider the problem of incrementally recomputing k_l after the deletion of an arbitrary node v. An idea is using again Equations 3 and 4, and the incremental algorithm developed in Paragraph 3, observing that:

i. For any induced subgraph F of G involved by Equation 4, $k_m^F(v) = 0$,

ii. For any auxiliary graph $A^{F,p}$ involved by Equation 3, we need to recompute $k_3^{A^{F,p}}$ after the deletion of the nodes in $S^{F,p}(v)$.

Consider an auxiliary graph $A^{F,p}$ from which we want to delete the nodes in $S^{F,p}(v)$. Let us associate a data structure $DS^{F,p}$ of the kind described in Paragraph 3 to $A^{F,p}$ and let $Z^{F,p}$ be its component Z. The initialization of $DS^{F,p}$ costs $O(n^{\omega p})$. Using the incremental algorithm of Paragraph 3, deleting the $O(n^{p-1})$ nodes of $S^{F,p}(v)$ from $A^{F,p}$, which contains $O(n^p)$ nodes, costs $O(n^{p-1}(n^{p(1+\varepsilon)} + n^{p(\omega(1,\varepsilon,1)-\varepsilon)}))$ for any $\varepsilon \in [0,1]$. Notice that the optimal value of ε again satisfies Equation 2 (it does not depend on p).

We can do better for high values of p, considering the particular structure of the problem. As the nodes in $S^{F,p}(v)$ are not connected, the triangles removed from $A^{F,p}$ deleting distinct elements of $S^{F,p}(v)$ are distinct too. Moreover we need the value of $C^{F,p}$ only after the deletion of all the nodes in $S^{F,p}(v)$. We can then update $C^{F,p}[s]$, for each $s \in V(A^{F,p})$, in a single step:

$$C^{F,p}[s] = \begin{cases} 0 & \text{if } s \in S^{F,p}(v), \\ C^{F,p}[s] - \sum_{u \in S^{F,p}(v)} P^{F,p}[s,u] \cdot D^{F,p}[s,u] & \text{otherwise,} \end{cases}$$

where $D^{F,p}$ and $P^{F,p}$ are updated after $C^{F,p}$ ($L^{F,p}$ is not used). With this approach, the updating cost is $O(n^{p\omega(1,\frac{p-1}{p},1)})$, that is $O(n^{\omega p - \frac{\omega-2}{1-\alpha}})$ according to Theorem 2. Under the reasonable assumption that the function $(\omega(1,r,1) - r)$, for $r \in [0,1]$, is not increasing in r, this complexity is smaller than the previous one for $\varepsilon < \frac{p-1}{p}$:

$$p\omega(1,\frac{p-1}{p},1) = p(\omega(1,\frac{p-1}{p},1) - \frac{p-1}{p}) + p - 1 \leq p - 1 + p(\omega(1,\varepsilon,1) - \varepsilon).$$

Thus we can update the number of triangles in which each node of the $O(n^q)$ auxiliary graphs is contained in $O(n^q T(n,p,\varepsilon))$ time for any $\varepsilon \in [0,1]$, where $T(n,p,\varepsilon) = \min\{n^{p-1}(n^{p(1+\varepsilon)} + n^{p(\omega(1,\varepsilon,1)-\varepsilon)}), n^{p\omega(1,\frac{p-1}{p},1)}\})$, and then recompute the function k_l within the same time bound, using Equations 3 and 4.

Theorem 3. *There is an incremental procedure to update the number of l-cliques, $l \geq 3$, in which each node of a graph is contained, after the deletion of an arbitrary node, which has an $O(n^{p\omega+q})$ initialization cost and an $O(n^q T(n,p,\varepsilon))$ updating cost for any $\varepsilon \in [0,1]$, where n is the number of nodes in the original graph, $p = \lfloor \frac{l}{3} \rfloor$, $q = l$ (mod 3), and $T(n,p,\varepsilon) = \min\{n^{p-1}(n^{p(1+\varepsilon)} + n^{p(\omega(1,\varepsilon,1)-\varepsilon)}), n^{p\omega(1,\frac{p-1}{p},1)}\})$.*

The current best bounds on $\omega(1,r,1)$ imply an $O(n^{2.575p+q-1})$ updating cost for $3 \leq l \leq 8$ and an $O(n^{2.376p+q-0.532})$ updating cost for $l \geq 9$.

5 Fast l Inverse Consistency

In this section we present an application of our results to an important family of constraint programming filtering techniques. A binary constraint network consists of a finite set X of variables, defined on finite domains, and a finite set C of binary constraints. Let k be the number of variables and d the size of the largest domains. A value assignment is a pair (x,w), where x is a variable and w is a value in the domain of x. A binary constraints $C_{\{x,y\}}$ describes the compatible pairs of value assignments for variables x and y, $x \neq y$ (we assume that two distinct value assignments relative to the same variable are not compatible). An instantiation of X is a set I of value assignments such that for each variable $x \in X$ there is exactly one value assignment $(x,w) \in I$. An instantiation I of X satisfies a constraint $C_{\{x,y\}}$ if and only if the 2 value assignments in I corresponding to x and y are compatible according to $C_{\{x,y\}}$. The binary constraint satisfaction problem consists in deciding if an instantiation of X exists which satisfies all the constraints in C. Each instantiation of this kind is called a solution for the binary constraint network. A binary constraint network can be represented between a k-partite graph G, which has a node for each value assignment and an edge between a pair of nodes if and only if the corresponding value assignments are compatible (to each partition correspond the value assignments relative to the same variable). To each solution corresponds a k-clique in G. The complexity of finding a k-clique in G can be greatly reduced in practice removing from G the nodes not satisfying a given property which all the nodes in a k-clique need to satisfy. One of the most interesting properties of this kind is l *inverse consistency*, $2 \leq l \leq k$ (Freuder & Elfe 1996, Mackworth 1977). Let g_l be the set of subgraphs of G induced by the set of nodes contained in l distinct partitions. Let moreover $g_l(v)$ be the subset of g_l formed by the subgraphs containing node v:

$$g_l(v) = \{G' \in g_l : v \in V(G')\}.$$

A node v is l Inverse Consistent (*l-IC*) if and only if it is contained in at least one l-clique in all the subgraphs $G' \in g_l(v)$, that is if and only if:

$$\forall G' \in g_l(v) : k_l^{G'}(v) > 0. \tag{5}$$

A graph is *l-IC* if and only if all its nodes are *l-IC*. Consider the problem of computing the largest *l-IC* induced subgraph of G, $ic_l(G)$. This problem is well defined as $ic_l(G)$ is unique:

Theorem 4. *The graph $ic_l(G)$ is unique.*

Proof: Suppose that there exist two distinct largest *l-IC* induced subgraphs of G, G' and G''. Then $G''' = G[V(G') \bigcup V(G'')]$ is an *l-IC* induced subgraph of G larger than G' and G'', that is a contradiction. $\square$

Clearly $ic_l(G)$ contains all the k-cliques of G, but it can be much smaller (and thus finding a k-clique in it can be much easier). The fastest algorithms known to compute $ic_2(G)$ and $ic_3(G)$ have an $O(k^2d^2)$ and an $O(k^3d^3)$ time complexity respectively (Bessiere 1994, Debruyne 2000). The approach of (Debruyne 2000) can be easily generalized to compute $ic_l(G)$ in $O(k^ld^l)$ time for $l > 3$ and, to the best of our knowledge, no asymptotically faster algorithms are known for $l > 3$.

A straightforward consequence of the incremental algorithms introduced in Paragraph 4, is a faster algorithms to compute $ic_l(G)$ for $l \geq 3$. We initialize all the data structures needed to incrementally recomputing the function $k_l^{G'}$, for each $G' \in g_l$, according to the procedure described in Paragraph 4, and we store the initial value of $V(G)$ in a set V'. Then, for each node $v \in V'$, we check if v satisfies Property 5. If not, we execute the following deletion procedure on v:

i. We remove v from V',

ii. We insert v in a set *DSet*, initially empty,

iii. We compute the new value of $k_l^{G'}$, for all $G' \in g_l(v)$, after the deletion of v.

A deletion can transform an *l-IC* node in a not *l-IC* one. We then need to propagate the effects of deletions: until *DSet* is not empty, we extract a node u from *DSet* and we check, for all the subgraphs $G' \in g_l(u)$ and for each node $v \in V(G') \bigcap V'$, if v is contained in at least one l-clique of $G'[V']$ (if and only if $k_l^{G'}(v) > 0$). If not, v is no more *l-IC* and we execute on v the deletion procedure above described. At the end of the process, $G[V']$ is equal to $ic_l(G)$. The process can be stopped earlier if a partition becomes empty: in that case, in fact, $ic_l(G)$ is the empty graph and we need not to remove the remaining nodes explicitly.

The initialization of all the data structures costs $O(k^l d^{\omega p+q})$. As each $G' \in g_l$ is interested by at most $O(d)$ deletions ($|V(G')| \leq ld$), the global cost to recompute the $k_l^{G'}$s is $O(k^l d^{q+1} T(d,p,\varepsilon))$ for any $\varepsilon \in [0,1]$, where $T(d,p,\varepsilon) = \min\{d^{p-1}(d^{p(1+\varepsilon)} + d^{p(\omega(1,\varepsilon,1)-\varepsilon)}), d^{p\omega(1,\frac{p-1}{p},1)}\})$. This is also an upper bound on the global cost of the algorithm.

Theorem 5. *There is an algorithm to compute $ic_l(G)$, $l \geq 3$, with an $O(k^l d^{q+1} T(d,p,\varepsilon))$ time complexity for any $\varepsilon \in [0,1]$, where $p = \lfloor \frac{l}{3} \rfloor$, $q = l \pmod 3$, and $T(d,p,\varepsilon) = \min\{d^{p-1}(d^{p(1+\varepsilon)} + d^{p(\omega(1,\varepsilon,1)-\varepsilon)}), d^{p\omega(1,\frac{p-1}{p},1)}\}$.*

The current best bounds on $\omega(1,r,1)$ imply an $O(k^l d^{2.575p+q})$ cost for $3 \leq l \leq 8$ and an $O(k^l d^{2.376p+q+0.468})$ cost for $l \geq 9$. This improves on the $O(k^l d^l)$ bound for every $l \geq 3$.

Acknowledgements

Thanks to Friedrich Eisenbrand and Paolo Ventura for their valuable suggestions.

References

Alon, N., Yuster, R. & Zwick, U. (1997), 'Finding and counting given length cycles', *Algorithmica* **17**(3), 209–223.

Bessiere, C. (1994), 'Arc-consistency and arc-consistency again', *Artificial Intelligence* **65**, 179–190.

Chiba, N. & Nishizeki, T. (1985), 'Arboricity and subgraph listing algorithms', *SIAM J. Comput.* **14**, 210–233.

Coppersmith, D. & Winograd, S. (1990), ‘Matrix multiplication via arithmetic progressions’, *J. Symbolic Comput.* **9**, 251–280.

Debruyne, R. (2000), A property of path inverse consistency leading to an optimal pic algorithm, *in* ‘Proc. of ECAI-00’, Berlin, Germany, pp. 89–92.

Demetrescu, C. & Italiano, G. (2000), Fully dynamic transitive closure: breaking through the $o(n^2)$ barrier, *in* ‘Proc. of the 41th Annual Symposium on Foundations of Computer Science (FOCS’00)’, pp. 381–389.

Freuder, E. & Elfe, D. (1996), Neighborhood inverse consistency preprocessing, *in* ‘Proc. of AAAI-96’, Portland, OR, pp. 202–208.

Huang, X. & Pan, V. (1998), ‘Fast rectangular matrix multiplication and applications’, *Journal of Complexity* **14**, 257–299.

Itai, A. & Rodeh, M. (1978), ‘Finding a minimum circuit in a graph’, *SIAM J. Comput.* **7**, 284–304.

Mackworth, A. (1977), ‘Consistency in networks of relations’, *Artificial Intelligence* **8**, 99–118.

Mehlhorn, K. (2000), Constraint programming and graph algorithms, *in* ‘Proc. of ICALP 2000, Lecture Notes in Computer Science’.

Nesetril, J. & Poljak, S. (1985), ‘On the complexity of the subgraph problem’, *Commentationes Mathematicae Universitatis Carolinae* **14**, 415–419.

Zwick, U. (1998), All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms, *in* ‘Proc. of the 39th IEEE Annual Symposium on Foundations of Computer Science (FOCS’98)’, pp. 310–319.

MPI-I-2002-4-001	M. Goesele	Tutorial Notes ACM SM 02 A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2002-2-008	W. Charatonik, J. Talbot	Atomic Set Constraints with Projection
MPI-I-2002-2-007	W. Charatonik, H. Ganzinger	Symposium on the Effectiveness of Logic in Computer Science in Honour of Moshe Vardi
MPI-I-2002-1-008	P. Sanders, J.L. Träff	The Factor Algorithm for All-to-all Communication on Clusters of SMP Nodes
MPI-I-2002-1-002	F. Grandoni	Incrementally maintaining the number of l-cliques
MPI-I-2002-1-001	T. Polzin	Using (sub)graphs of small width for solving the Steiner problem
MPI-I-2001-4-005	H.P.A. Lensch, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2001-4-004	S.W. Choi, H. Seidel	Linear One-sided Stability of MAT for Weakly Injective Domain
MPI-I-2001-4-003	K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel	Efficient Light Transport Using Precomputed Visibility
MPI-I-2001-4-002	H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web
MPI-I-2001-4-001	H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, H. Seidel	Image-Based Reconstruction of Spatially Varying Materials
MPI-I-2001-2-006	H. Nivelle, S. Schulz	Proceeding of the Second International Workshop of the Implementation of Logics
MPI-I-2001-2-005	V. Sofronie-Stokkermans	Resolution-based decision procedures for the universal theory of some classes of distributive lattices with operators
MPI-I-2001-2-004	H. de Nivelle	Translation of Resolution Proofs into Higher Order Natural Deduction using Type Theory
MPI-I-2001-2-003	S. Vorobyov	Experiments with Iterative Improvement Algorithms on Completely Unimodel Hypercubes
MPI-I-2001-2-002	P. Maier	A Set-Theoretic Framework for Assume-Guarantee Reasoning
MPI-I-2001-2-001	U. Waldmann	Superposition and Chaining for Totally Ordered Divisible Abelian Groups
MPI-I-2001-1-007	T. Polzin, S. Vahdati	Extending Reduction Techniques for the Steiner Tree Problem: A Combination of Alternative-and Bound-Based Approaches
MPI-I-2001-1-006	T. Polzin, S. Vahdati	Partitioning Techniques for the Steiner Problem
MPI-I-2001-1-005	T. Polzin, S. Vahdati	On Steiner Trees and Minimum Spanning Trees in Hypergraphs
MPI-I-2001-1-004	S. Hert, M. Hoffmann, L. Kettner, S. Pion, M. Seel	An Adaptable and Extensible Geometry Kernel
MPI-I-2001-1-003	M. Seel	Implementation of Planar Nef Polyhedra