



## A Polynomial Delay Algorithm for Generating Connected Induced Subgraphs of a Given Cardinality

*Khaled Elbassioni*<sup>1</sup>

<sup>1</sup>Department of Electrical Engineering and Computer Science,  
Masdar Institute of Science and Technology, P.O.Box 54224, Abu Dhabi, UAE

### Abstract

We give a polynomial delay algorithm, that for any graph  $G$  and positive integer  $k$ , enumerates all connected induced subgraphs of  $G$  of order  $k$ . Our algorithm enumerates each subgraph in at most  $O((k \min\{n - k, k\Delta\})^2(\Delta + \log k))$  and uses linear space  $O(n + m)$ , where  $n$  and  $m$  are respectively the number of vertices and edges of  $G$  and  $\Delta$  is the maximum degree.

Submitted: January 2014	Reviewed: April 2015	Revised: April 2015	Accepted: April 2015	Final: May 2015
		Published: May 2015		
	Article type: Regular paper	Communicated by: P. Bose		

## 1 Introduction

Let  $G = (V, E)$  be an undirected graph of  $|V| = n$  vertices and  $m = |E|$  edges. Given an integer  $k \in \mathbb{Z}_+$ , we consider the problem  $\text{GEN}(G; k)$  of enumerating (or generating) all subsets  $X \subset V$  of vertices such that  $|X| = k$  and the subgraph  $G[X]$  induced on  $X$  is connected. Let us denote the family of such vertex sets by  $\mathcal{C}(G; k)$ .

Typically, the size of  $\mathcal{C}(G; k)$  is exponentially large in  $k$ . In fact, it was shown in [10] that there exists a family of connected graphs  $G$  with maximum degree  $\Delta < \frac{2n}{k}$ , for which  $|\mathcal{C}(G; k)| \geq n(\frac{\Delta}{2})^{k-1}$ . On the other hand, an upper bound of  $\frac{(\epsilon\Delta)^k}{(\Delta-1)^k}$  on the size of  $\mathcal{C}(G; k)$  was also given in [10].

An enumeration algorithm for  $\mathcal{C}(G; k)$  is said to be *output (or total) polynomial* if the algorithm outputs all the elements of  $\mathcal{C}(G; k)$  in time polynomial in  $n$  and  $|\mathcal{C}(G; k)|$ . Avis and Fukuda [1] introduced the *reverse search method* for enumeration, and used it to solve, among several other problems, the problem of enumerating all connected induced subgraphs of size *at most*  $k$ . Uehara [10] noted that such an algorithm is total polynomial for enumerating  $\mathcal{C}(G; k)$  when  $k$  is a fixed constant. In fact, since the algorithm of [1] enumerates the families  $\mathcal{C}(G; i)$  for all  $1 \leq i \leq k$ , the total the running time of the algorithm is bounded by (see [10]):

$$O\left(n + m + \sum_{i=1}^{k-1} |\mathcal{C}(G; i)| + k^2 |\mathcal{C}(G; k)|\right), \quad (1)$$

which is upper bounded by  $O\left(n + m + \frac{(\epsilon\Delta)^k}{(\Delta-1)} k^2\right)$ , using the upper bound on  $|\mathcal{C}(G; k)|$  mentioned above.

However, we note that such a bound is not polynomial, when  $k$  is part of the input<sup>1</sup>. In fact, considering the lower bound example mentioned above, we observe that, when  $k = n - \Delta$ , the size of  $\mathcal{C}(G; k)$  is at most  $\binom{n}{k} < n^\Delta$ , while for all  $i < \frac{2n}{\Delta}$ ,  $|\mathcal{C}(G; i)| \geq n(\frac{\Delta}{2})^{i-1}$ . Thus for  $i = \frac{2n}{\Delta} - 1$ , the total running time in (1) will be at least  $n(\frac{\Delta}{2})^{\frac{2n}{\Delta}-2}$ . Setting, for instance,  $\Delta = n^\epsilon$  for some  $\epsilon \in (0, \frac{1}{2})$ , and assuming  $n$  is large enough, we get that  $i = 2n^{1-\epsilon} - 1 < k = n - n^\epsilon$ , and thus the running time in (1) is at least

$$n^{(2n^{1-\epsilon}-2)(\epsilon-\frac{1}{\log n})+1} > |\mathcal{C}(G; k)|^{(2n^{1-2\epsilon}-2n^{-\epsilon})(\epsilon-\frac{1}{\log n})} = |\mathcal{C}(G; k)|^{n^{\Omega(1)}}.$$

Thus, the algorithm suggested in [10] is *not* a total polynomial algorithm.

An enumeration algorithm for  $\text{GEN}(G; k)$  is said to have a *polynomial delay* of  $p(n, m, k)$  if it outputs all the elements of  $\mathcal{C}(G; k)$ , such that the running time between any two successive outputs is at most  $p(n, m, k)$ . The algorithm is said

<sup>1</sup>An analogy can be drawn from the enumeration of face lattices of polytopes, where enumerating all faces of dimension at most  $k$  is solvable in total polynomial time, while enumerating faces of dimension exactly  $k$  includes the well-known *vertex enumeration problem*, whose complexity is an outstanding open question. This discrepancy is due to the fact that, there are polytopes (the so-called *fat-lattice* polytopes, see e.g. [9, 4]) where the total number of faces is exponentially large in the number of facets and vertices

to use *linear space*, if the total space required by the algorithm (excluding the space for writing the output) is  $O(n + m)$ . Recently, Karakashian et al. [7] gave an algorithm with delay  $O(\Delta^k)$  for solving  $GEN(G; k)$ .

In this short note, we give a bound that is *polynomial* in  $k$ .

**Theorem 1** *There is an algorithm for solving  $GEN(G; k)$ , for any graph  $G = (V, E)$  and integer  $k \in \mathbb{Z}_+$ , with polynomial delay  $O((k \min\{(n - k), k\Delta\})^2(\Delta + \log k))$  and space  $O(n + m)$ .*

We remark that the polynomial delay bound can be improved if we do not insist on using polynomial space (that is, the space used by the algorithm may depend polynomially on  $|\mathcal{C}(G; k)|$ ); see Corollary 1 below.

Our proof of Theorem 1 is also based on using the the reverse search method [1]. In fact we will consider more generally the *supergraph method* for enumeration, which can be thought of as a generalization of the reverse search method. This method will be briefly explained in the next section. Then we prove Theorem 1 in Sections 3 and 4.

The problem of enumerating of connected induced subgraphs of size  $k$  arises in several applications, such as keyword search over RDF graphs in Information Retrieval, and consistency analysis in Constrained Processing; see, e.g., [5, 7] and the references therein.

## 2 The Supergraph Approach

This technique generally works by building and traversing a directed (super)graph  $\mathcal{G} = (\mathcal{C}(G; k), \mathcal{E})$ , defined on the family  $\mathcal{C}(G; k)$ . The arcs of  $\mathcal{G}$  are defined by a *neighborhood function*  $\mathcal{N} : \mathcal{C}(G; k) \rightarrow 2^{\mathcal{C}(G; k)}$ , that to any  $X \in \mathcal{C}(G; k)$  assigns a set of its successors  $\mathcal{N}(X)$  in  $\mathcal{G}$ . A special node  $X_0 \in \mathcal{C}(G; k)$  is identified from which all other nodes of  $\mathcal{G}$  are reachable. The algorithm works by traversing, either, in *depth-first* or *breadth-first* search order, the nodes of  $\mathcal{G}$ , starting from  $X_0$ . If  $\mathcal{G}$  is *strongly connected* then  $X_0$  can be any node in  $\mathcal{C}(G; k)$ .

To avoid confusion, in the following, we will distinguish the vertices of  $G$  and  $\mathcal{G}$  by referring to them as *vertices* and *nodes*, respectively.

The following fact is known about this approach (see e.g. [1, 2, 6, 8]):

**Proposition 1** *Consider the supergraph  $\mathcal{G}$  and suppose that*

- (i)  $\mathcal{G}$  is strongly connected;
- (ii) a node  $X_0$  in  $\mathcal{G}$  can be found in time  $t_0(n, m, k)$ ;
- (iii) for any node  $X$  in  $\mathcal{G}$ ,  $|\mathcal{N}(X)| \leq N(n, m, k)$  and we can generate  $\mathcal{N}(X)$  with delay  $t(n, m, k)$ ;

*then  $\mathcal{C}(G; k)$  can be generated with delay  $O(\max\{t_0(n, m, k), (t(n, m, k) + \log |\mathcal{C}(G; k)|) \cdot N(n, m, k)\})$  and space  $O(n + m + k|\mathcal{C}(G; k)|)$ . If instead of (i),*

(iv) there is a function  $f : \mathcal{C}(G; k) \setminus \{X_0\} \rightarrow \mathcal{C}(G; k)$ , that for every node  $X \neq X_0$  in  $\mathcal{G}$ , identifies (in a unique way), in time  $t_1(n, m, k)$ , a node  $X' = f(X)$  in  $\mathcal{G}$  such that  $X \in \mathcal{N}(X')$ , and  $f$  satisfies the following acyclicity property: there exist no  $X_{\ell+1} = X_1, X_2, \dots, X_\ell \in \mathcal{C}(G; k)$  such that  $X_i = f(X_{i+1})$  for  $i = 1, \dots, \ell$ ,

then  $\mathcal{C}(G; k)$  can be generated with delay  $O(\max\{t_0(n, m, k), (t(n, m, k) + t_1(n, m, k)) \cdot N(n, m, k)\})$  and space  $O(n + m)$ .

The proof is straightforward. For the first claim, we essentially traverse a *breadth-first* search (BFS) tree on  $\mathcal{G}$ , starting from node  $X_0$ . We maintain a *balanced binary search tree* (BST) on the elements generated, sorting them, say, according to some *lexicographic order*. We also keep a *queue* of all elements that have been generated but whose neighborhoods have not been yet explored. When processing a node  $X$  in the queue, we output  $X$  and then generate all its neighbors in  $\mathcal{G}$ , but only insert in the queue a neighbor  $X'$  if it has not been yet stored in the BST.

To achieve the second claim, we instead traverse a *depth-first* search (DFS) tree on  $\mathcal{G}$ , starting from node  $X_0$ . (This is essentially the reverse search method) When traversing a node  $X$ , we generate all neighbors of  $X$  in  $\mathcal{G}$ , but only proceed the search on a neighbor  $X'$ , if  $X = f(X')$  is the unique "parent" defined in (iv). The fact that  $f$  is a function satisfying the acyclicity property implies that all the nodes in  $\mathcal{G}$  are processed exactly once (since for any node  $X$  in  $\mathcal{G}$  there is a unique path leading to  $X_0$ ). In order to obtain the claimed delay bound, we output a node  $X$  just after the first visit to it, if the depth of  $X$  in the tree is odd (assume the root  $X_0$  has depth 1), or  $X$  is a leaf; otherwise,  $X$  is output just before coming back to the parent. (Note that this way of distributing the outputs over time ensures that the delay between two successive outputs is not more than the time to generate at most two nodes. Indeed, if  $X$  has odd depth or  $X$  is a leaf, then the last node output before  $X$  would be the closet ancestor of  $X$  at odd depth; if  $X$  has even depth, then the last node output before  $X$  would be the closest node with even depth (or a child leaf if no such node exists) on the right-most path descending from  $X$ .) Note that we do not need to store the history along any search path since the parent of any node can be generated efficiently.

Clearly, we may and will assume in the rest of the paper that the graph  $G$  is connected. In the next section, we will prove that this assumption implies that the supergraph  $\mathcal{G}$  in our case is strongly connected (and in fact has diameter  $\delta \leq n - k$ ). Let us now set the other parameters in Proposition 1, corresponding to our problem  $\text{GEN}(G; k)$ . We assume an order on the vertices of  $G$ , defined by an arbitrary DFS tree, which also naturally defines a lexicographic order " $\preceq$ " on the vertex sets. Clearly, the lexicographically smallest node  $X_0 \in \mathcal{C}(G; k)$  can be found in time  $t_0(n, m, k) = O(k\Delta)$  by traversing the DFS tree starting from the smallest vertex in  $G$  and processing vertices in DFS order until exactly  $k$  vertices are visited.

Given a node  $X \in \mathcal{C}(G; k)$ , we have, with the neighborhood definition given below,  $|\mathcal{N}(X)| \leq k \cdot \min\{(n - k), k\Delta\}$ , and each element in  $\mathcal{N}(X)$  can be generated in time<sup>2</sup>  $t(n, m, k) = O(k(\Delta + \log k))$ . These bounds, together with the upper bound  $|\mathcal{C}(G; k)| \leq \frac{(e\Delta)^k}{(\Delta - 1)^k}$  imply that the set  $\mathcal{C}(G; k)$  can be generated with polynomial delay  $O(k^2 \cdot \min\{(n - k), k\Delta\}(\Delta + \log k))$  and space  $O(n + m + k|\mathcal{C}(G; k)|)$ .

In order to achieve a polynomial space bound (independent of  $|\mathcal{C}(G; k)|$ ), we will need a stronger claim, namely that every node  $X \in \mathcal{C}(G; k) \setminus \{X_0\}$  is reachable from  $X_0$  by a *monotonically increasing* lexicographically ordered sequence of nodes. This claim will be proved for any DFS ordering on the vertices of  $G$ , and allows us to identify the parent  $X'$  of any node  $X$  by defining, for instance,  $X' = X \cup \{u\} \setminus \{v\}$ , where  $u \in V \setminus X$  and  $v \in X$  are respectively the smallest and largest vertices (in the DFS order) such that  $X' \prec X$  and the graph  $G[X \cup \{u\} \setminus \{v\}]$  is connected. Note that finding  $X'$  can be done in time  $t_1(n, m, k) \leq t(n, m, k) \cdot \min\{(n - k), k\Delta\}$ .

### 3 The Neighborhood Operator for $\mathcal{C}(G; k)$

For a set  $X \in \mathcal{C}(G; k)$ , it is natural to define the neighbors of  $X$  as those which are obtained from  $X$  by exchanging one vertex:

$$\mathcal{N}(X) = \{X' \in \mathcal{C}(G; k) : X \cap X' = k - 1\}.$$

It is worth comparing our neighborhood definition to the one suggested in [1] for generating all connected induced subgraphs of size *at most*  $k$ . In the latter definition, two sets  $X, X' \subseteq V$  are neighbors if they *differ* in exactly one vertex. The claim of strong connectivity follows immediately from the simple facts that if  $X \subset V$  is such that  $G[X]$  is connected then there is a vertex  $u \in X$  such that  $G[X \setminus \{u\}]$  is also connected, and similarly, there is a vertex  $u \in V \setminus X$  such that  $G[X \cup \{u\}]$  is connected.

### 4 Strong Connectivity

We prove first that the supergraph  $\mathcal{G}$  is strongly connected.

**Lemma 1** *Let  $X, Y$  be two distinct elements of  $\mathcal{C}(G; k)$ . Then there exist vertex sets  $X_1, X_2, \dots, X_\ell \in \mathcal{C}(G; k)$  such that  $X_1 = X$ ,  $X_\ell = Y$ ,  $\ell \leq n - k + 1$ , and for  $i = 1, \dots, \ell - 1$ ,  $X_{i+1} \in \mathcal{N}(X_i)$ .*

**Proof:** Let  $d(Z, Z')$  be the (shortest) distance between the two vertex sets  $Z, Z'$  in  $G$ . Suppose we have already constructed  $X_i$ . We consider two cases.

*Case 1.*  $d(X_i, Y) > 0$  (and hence  $X_i \cap Y = \emptyset$ ). Let  $u_0, u_1, \dots, u_r$  be the ordered sequence of vertices on the shortest path between  $X_i$  and  $Y$  in  $G$ , where  $u_0 \in X_i$

<sup>2</sup>This can be achieved by using a simple disjoint-set data structure; see e.g. Theorem 21.1 in [3]. We remark that this bound can be improved through the use of more sophisticated data structures.

and  $u_r \in Y$ . Let  $T$  be a spanning tree in  $G[X_i]$ . Then  $T$  has a leaf  $v \neq u_0$ . We define  $X_{i+1} = X_i \cup \{u_1\} \setminus \{v\}$ . By construction,  $X_{i+1} \in \mathcal{C}(G; k)$ , and  $d(X_{i+1}, Y) < d(X_i, Y)$ . Since  $d(X_1, Y) \leq n - 2k + 1$ , we will arrive at case 2 after at most  $n - 2k + 1$  iterations.

*Case 2.*  $d(X_i, Y) = 0$ . Then there exists a vertex  $z \in X_i \cap Y$ . Let  $C(X_i; z)$  be (the vertex set of) the connected component containing  $z$  in  $G[X_i \cap Y]$ . We claim that there exists a vertex set  $X_{i+1} \in \mathcal{N}(X_i)$  such that  $|C(X_{i+1}; z)| > |C(X_i; z)|$ . Indeed, let us contract  $C(X_i; z)$  into a single vertex  $w$  and denote the new graph by  $G'$ . Then, by the connectivity of  $G[Y]$ , there is an edge  $\{w, u\}$  in the graph  $G'[Y \cup \{w\} \setminus C(X_i; z)]$ , where  $u \in Y$  ( $u \neq w$ ). Similarly, the graph  $G'[X_i \cup \{w\} \setminus C(X_i; z)]$  is connected and hence has a spanning tree  $T$ . Let  $v \neq w$  be a leaf in  $T$ . Then  $X_{i+1} = X_i \cup \{u\} \setminus \{v\}$  satisfies the claim. This claim implies that in at most  $k - 1$  iterations of case 2, we will have  $X_{i+1} = Y$ .  $\square$

In view of Proposition 1, Lemma 1 implies that all the elements of  $\mathcal{C}(G; k)$  can be enumerated with polynomial delay.

**Corollary 1** *There is an algorithm for solving  $\text{GEN}(G; k)$ , for any graph  $G = (V, E)$  and integer  $k \in \mathbb{Z}_+$ , with polynomial delay  $O(k^2 \cdot \min\{(n - k), k\Delta\} \cdot (\Delta + \log k))$ .*

In order to prove that  $\text{GEN}(G; k)$  can be solved also with polynomial space, we need the following result.

**Lemma 2** *Consider the lexicographic ordering " $\preceq$ " on  $\mathcal{C}(G; k)$  defined by a DFS order on the vertices of  $G$ . Let  $X$  be any element in  $\mathcal{C}(G; k)$  that is not lexicographically smallest. Then there is an  $X' \in \mathcal{C}(G; k)$  such that  $X' \in \mathcal{N}(X)$  and  $X' \prec X$ .*

**Proof:** Let us denote the lexicographically smallest element of  $\mathcal{C}(G; k)$  by  $X_0$ . Since  $X_0 \prec X$ , it holds that  $w := \min_{u \in X_0 \setminus X} u < z := \min_{u \in X \setminus X_0} u$ . We use the following simple property of the DFS tree: for  $x < y$ , the DFS-tree walk (that is, the sequence of vertices visited on the way in the DFS order) from  $x$  to  $y$  contain only nodes  $z \leq y$ . We consider a number of cases:

*Case 1.*  $z$  is not a *cut-vertex* in  $G[X]$  (that is,  $G[X] - z$  is connected). We consider two subcases.

*Subcase 1.1.* The DFS-tree walk from  $w$  to  $z$  enters  $X$  at a vertex  $x \neq z$  through an edge  $\{y, x\}$ . Thus  $x \in X$ ,  $y \notin X$  and  $y < z$ . Since  $z$  is not a cut vertex, there is a spanning tree in  $G[X]$  which has  $z$  as a leaf. Then  $X' = X \cup \{y\} \setminus \{z\}$  satisfies the claim.

*Subcase 1.2.* The DFS-tree walk from  $w$  to  $z$  enters  $X$  at  $z$  through an edge  $\{y, z\}$ . If all vertices  $u \in X \setminus \{z\}$  satisfy  $u > z$ , then let  $x \neq z$  be a leaf in a spanning tree in  $G[X]$ , and set  $X' = X \cup \{y\} \setminus \{x\}$  which satisfies the claim. Otherwise, there is a vertex  $x$  in  $X \cap X_0$  that precedes  $z$  in the DFS order. Then  $x$  necessarily precedes  $w$  (otherwise we are in case 1.1, as the DFS walk from  $w$  to  $x$  would enter  $X$  through an edge  $\{u, v\}$ , where  $u < x < z$ ), and the DFS

walk between  $x$  and  $w$  must exit  $X$  through some edge  $\{u, v\}$  with  $u \in X$  and  $v \notin X$  such that  $v \leq w < z$ . Then setting  $X' = X \cup \{v\} \setminus \{z\}$  satisfies the claim.

*Case 2.*  $z$  is a *cut-vertex* in  $G[X]$ . Let  $X_1$  be the vertex set of the connected component in  $G[X] - z$  containing the smallest vertex  $u$  in  $X \setminus \{z\}$ , and let  $X_2$  be the vertex set of any other connected component in  $G[X] - z$ . Let  $v \neq z$  be a leaf in a spanning tree in  $G[X_2 \cup \{z\}]$ . We consider two subcases.

*Subcase 2.1.* The DFS-tree walk from  $u$  to  $v$  does not go through  $z$ . Then it must be the case that the walk leaves  $X_1$ , and hence  $X$ , through an edge  $\{x, y\}$ , where  $x \in X_1$  and  $y \notin X$ . Since  $y < v$ , the set  $X' = X \cup \{y\} \setminus \{v\}$  satisfies the claim.

*Subcase 2.2.* The DFS-tree walk from  $u$  to  $v$  does go through  $z$ . Then  $z < v$  and the DFS walk from  $w$  to  $z$  enters  $X$  at a vertex  $x \neq v$  through an edge  $\{y, x\}$ , where  $y \notin X$ . Since  $y < z < v$ , we can set  $X' = X \cup \{y\} \setminus \{v\}$  to satisfy the claim.  $\square$

We note that the conclusion of Lemma 2 is not true if we replace the DFS order of the vertices by an arbitrary order. Consider for instance, the case when  $G$  is a path with vertices numbered  $1, 2, \dots, k, n, k+1, \dots, n-1$ , in the order they appear on the path, where  $n > 2k+1$ . Then the set  $X = \{k+1, k+2, \dots, 2k\}$  has two neighbors, namely,  $X' = \{n, k+1, k+2, \dots, 2k-1\}$  and  $X'' = \{k+2, k+3, \dots, 2k+1\}$ , both of which are lexicographically larger than  $X$ .

## Acknowledgements

I thank Shady Elbassuoni for bringing this problem to my attention, and the anonymous reviewer for the careful reading and many valuable remarks.

## References

- [1] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996. doi:10.1016/0166-218X(95)00026-N.
- [2] E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. Generating vertices of polyhedra and related problems of monotone generation. In D. Avis, D. Bremner, and A. Deza, editors, *Proceedings of the Centre de Recherches Mathématiques at the Université de Montréal, special issue on Polyhedral Computation (CRM Proceedings and Lecture Notes)*, volume 49, pages 15–43, 2009.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- [4] M. E. Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8(3):381–402, 1983. doi:10.2307/3689308.
- [5] S. Elbassuoni and R. Blanco. Keyword search over RDF graphs. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 237–242, 2011. doi:10.1145/2063576.2063615.
- [6] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.
- [7] S. Karakashian, B. Y. Choueiry, and S. G. Hartke. An algorithm for generating all connected subgraphs with  $k$  vertices of a graph., 2013. Manuscript. URL: <http://www.math.unl.edu/~shartke2/math/papers/k-subgraphs.pdf>.
- [8] B. Schwikowski and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253–265, 2002. doi:10.1016/S0166-218X(00)00339-5.
- [9] A. Shioura, A. Tamura, and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3):678–692, 1997. doi:10.1137/S0097539794270881.
- [10] R. Uehara. The number of connected components in graphs and its applications. Manuscript. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.5362>.