# Reporting maximal cliques: new insights into an old problem

Frédéric Cazals, Chinmay Karande

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Reporting maximal cliques: new insights into an old problem*

Frédéric Cazals — Chinmay Karande

**N° 5615**

Juin 2005

Thème SYM


ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

# Reporting maximal cliques: new insights into an old problem

Frédéric Cazals [*] , Chinmay Karande [†]

Thème SYM — Systèmes symboliques
Projets Geometrica

Rapport de recherche n° 5615 — Juin 2005 — 24 pages

**Abstract:** Reporting maximal cliques of a graph is a fundamental problem arising many areas. Surprisingly, this problem is often tackled resorting to the old Bron-Kerbosch algorithm (1973), or its recent variant by I. Koch (2001). Both algorithms suffer from a poor output sensitivity and worst-cases.

In this context, this paper makes three contributions. First, we show that a slight modification of the greedy pivoting strategy used by I. Koch allows one to get rid of worst-cases, also improving overall performances. Second, exploiting the recursive structure of non maximal cliques, we show the pivoting strategy developed by I. Koch is a particular case of a more general optimization strategy based on the concept of *dominated* nodes. Using different instantiations of this concept, we design four modified Bron-Kerbosch algorithms, with better output-sensitivity. Third, we discuss implementation issues and provide a detailed experimental study on random graphs.

The bottom-line of this study is the investigation of the trade-off between output sensitivity and the overhead associated to the identification of dominated nodes.

**Key-words:** Maximal cliques, Shape Matching

[*] INRIA Sophia-Antipolis, Geometrica project
[†] IIT Bombay, India; ckarande@gmail.com.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

# Calcul des cliques maximales: une cure de jouvence

**Résumé :** Le calcul de toutes les cliques maximales d'un graphe est un problème fondamental qui se rencontre dans nombre de domaines. De façon surprenante, ce calcul est généralement fait en utilisant l'algorithme de Bron-Kerbosch (1973), où sa variante récente par I. Koch (2001). Malheureusement, ces deux algorithmes ont des propriétés médiocres de sensibilité à la sortie, et souffrent de cas pathologiques.

Dans ce contexte, ce travail propose trois contributions. Tout d'abord, nous proposons une modification de la stratégie du pivot utilisée par I. Koch de façon à éradiquer les cas pathologiques. De façon intéressante, cette stratégie permet aussi d'améliorer les performances globales de l'algorithme. Ensuite, en exploitant la structure récursive des cliques, nous montrons que la stratégie du pivot est un cas particulier d'une stratégie plus générale basée sur le concept des noeuds *dominés*. Ce concept peut être instantié de diverses façons pour améliorer la sensibilité à la sortie. Enfin, nous discutons diverses implémentations, et comparons celles-ci au gré d'une étude expérimentale portant sur des graphes aléatoires.

La ligne force de ce travail réside dans la compréhension des compromis opposant la sensibilité à la sortie et les coûts induits par l'amélioration de celle-ci.

**Mots-clés :** Graphes complets, Cliques Maximales, *Shape Matching*

# 1 Reporting maximal cliques

## 1.1 Maximum weight clique vs. maximal cliques.

Reporting maximal cliques of a graph is a fundamental problem arising wherever graphs are the natural way to model objects and their interactions. Example applications can be found in network design and analysis, social sciences, or mathematical biology –see the discussion below for illustrations on this later vein.

Reporting *all* maximal cliques of a graph should be not confused with the problem of finding the *maximum weight clique*, a well studied problem of combinatorial optimization for which a vast literature exist [BBPP99]. As opposed to the maximum weight clique problem, the output of clique detection algorithms may be exponentially sized —Fig. 1, so that an algorithm with provably good absolute running time is not possible. However, even with this consideration, any algorithm reporting all maximal cliques should, in some sense, output sensitive.

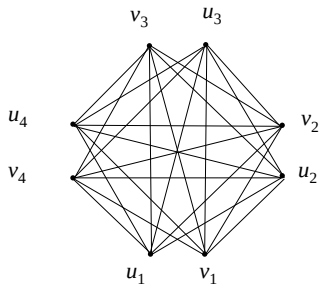


Figure 1: A graph with an exponential number of cliques

This said, to the best of our knowledge, the problem of reporting all maximal cliques of a graph is tackled resorting to the old Bron-Kerbosch algorithm [BK73] —denoted BK, or its recent variant by Ina Koch [Koc01]. Both algorithm suffer from a poor output sensitivity and worst-cases. To be precise, Bron-Kerbosch algorithm knows that a clique is non-maximal only when it reaches the clique. Hence, the algorithm enumerates an exponential number of cliques —in the output size, when the output consists of only a small number of them.

## 1.2 Contributions

As just outlined, an important measure for the BK algorithm is its output sensitivity. On the other hand, improving the output sensitivity may increase the complexity of the algorithm. In this context, the contribution of this paper is to investigate the structure of maximal cliques so as to develop strategies achieving a compromise between the output sensitivity and the overall complexity. As we shall see, the strategy developed, based on the notion of redundant nodes, also fudges around the worst-cases of the Bron-Kerbosch and I. Koch algorithms.

### 1.3 Applications

Our interest in the problem stems from computational biology. In particular, the detection of maximal cliques is fundamental for the question of *combinatorial partial shape matching* which consists, given two graphs, of reporting either all *Maximal Common Induced Subgraphs* —MCIS, or all *Maximal Common Edge Subgraphs* —MCES. See [Koc01, CK05a, CK05b] for a general discussion of these problems. Example applications of clique detection algorithms for the identification of MCIS and MCES in computational structural biology are the following. In [SBK92], MCIS involving four atoms are used to define rigid motions for docking a ligand into a protein; also for docking, MCIS involving pairs of compatibles atoms (each pair consisting of a pairs of (donor, acceptor) of electrons) are sought in [GWA00]; tertiary structure similarity is investigated from Maximum Common Induced Subgraph in [GARW93]; the detection of cliques for structure prediction is carried out in [SM98].

Other applications in this realm involve the analysis of metabolic networks [KWF01] —a problem likely to be of increasing interest in the systems biology era.

### 1.4 Notations and conventions

The size of a finite set $S$ is dented $\mid S \mid$. We shall denote the graph $G = (V[G], E[G])$, with $\mid V[G] \mid = n$. Given a node $u \in G$, $N[u]$ denotes the neighbors of $u$, i.e. $N[u] = \{v \mid (u,v) \in E[G]\}$. When pseudo-code is presented, arguments are passed by value —i.e. there are no side effects upon recursive calls.

### 1.5 Paper overview

Standard algorithms are discussed in section 2. Section 3 presents a pivoting strategy getting rid of worst-cases. The concept of dominated nodes is discussed in section 4, and a detailed experimental study is presented in section 5.

## 2 Standard algorithms

### 2.1 The original Bron-Kerbosch algorithm

To begin with, we recall the Bron-Kerbosch (BK) Algorithm. The algorithm maintains three sets of nodes $R, P, X$. Set $R$ stands for the currently growing clique; set $P$ stands for prospective nodes which are connected to all nodes in $R$ and using which $R$ can be expanded; set $X$ stands for eXcluded nodes, i.e. nodes which are not allowed to expand $R$ (All maximal cliques containing them have already been reported). An important invariant is that all nodes which are connected to $R$ (i.e. to every node of $R$) are either in $P$ or $X$. Set $R$ is expanded by prospectives nodes in $P$. To avoid reporting the same maximal clique several times, the algorithm performs the update $P = P - \{u_i\}$. And to avoid reporting cliques which are not maximal, the algorithm checks whether set $X$ is empty : if $X$ is non empty, the nodes in $X$ may be added to $R$, but this would yield previously found maximal cliques. See Fig. 2 for the pseudo-code.

Naturally, BK algorithm is exponential in the worst case. Since the algorithm knows that a clique is non-maximal only when it reaches it, all cliques are enumerated — an exponential number of them it the number of maximal cliques, when the output consists of only a small number of them.

```
Algorithm call: BK(∅, V[G], ∅).

BK(R, P, X)
 1: {returns all R_max which are maximal cliques such
    that R ⊂ R_max, R_max ∩ X = ∅ and R_max ⊆ R ∪ P.}
 2: if P = ∅ and X = ∅ then
 3:     Report R as a maximal clique
 4: else
 5:     Assume P = {u_1, u_2, ..., u_k}
 6:     for i ← 1 to k do
 7:         P = P − {u_i}
 8:         R_new = R ∪ {u_i}
 9:         P_new = P ∩ N[u_i]
10:         X_new = X ∩ N[u_i]
11:         BK(R_new, P_new, X_new)
12:         X = X ∪ {u_i}
```

Figure 2: Bron-Kerbosch Algorithm

```
Algorithm call: IK_*(∅, V[G], ∅).

IK_*(R, P, X)
 1: {returns all R_max which are maximal cliques such
    that R ⊂ R_max, R_max ∩ X = ∅ and R_max ⊆ R ∪ P.}
 2: if P = ∅ and X = ∅ then
 3:     Report R as a maximal clique
 4: else
 5:     Let u_p be the pivot vertex chosen from P.
 6:     Assume P = {u_1, u_2, ..., u_k}
 7:     for i ← 1 to k do
 8:         if u_i is not a neighbor of u_p then
 9:             P = P − {u_i}
10:             R_new = R ∪ {u_i}
11:             P_new = P ∩ N[u_i]
12:             X_new = X ∩ N[u_i]
13:             IK_*(R_new, P_new, X_new)
14:             X = X ∪ {u_i}
```

Figure 3: Bron-Kerbosch Algorithm with recognition of equal subtrees via pivot selection. Suffix _$*$ indicates the pivot selection strategy is not specified.

## 2.2 Optimizations by I. Koch

**Skipping the neighbors of a pivot.** An efficient heuristic to reduce the recursion tree for Bron-Kerbosch algorithm is analyzed by [Koc01]. (The analysis is carried out in that paper, although the heuristic is mentioned in applied papers such as [SBK92, GARW93].) The heuristic is based on identification and elimination of equal subtrees appearing in different branches of the algorithm. This identification is performed via the choice of a pivoting node $u_p \in P$. For convenience, given the pivot $u_p$, we shall decompose $P$ as $P = P^+ \cup P^-$ with: $P^- = P \cap N[u_p]$, the neighbors of the pivot in $P$ and $P^+ = P \backslash P^-$, the remaining nodes which are not neighbors of $P$. Note that by definition, a node cannot have edge to itself, hence $u_p$ belongs to $P^+$. Set $P^-$ is called sterile since no subset of $P^-$ can be appended to $R$ by itself to form a maximal clique, and hence these nodes can be skipped from the main `for` loop of the BK algorithm. Set $P^+$ is called fertile since each maximal clique that can be formed in the recursion subtree rooted at current point, must contain at least one node from

$P^+$. The pseudocode of this algorithm is presented on Fig. 3. This code is templated in the sense that the precise way the pivot is chosen is not described, whence the $\_*$ suffix.

The correctness of the above heuristic can be argued by visualizing a certain order in which vertices in $P$ are iterated over. The non-neighbors of the pivot vertex $u_p$ are selected first, followed by $u_p$ and the neighbors of $u_p$. However, by the time the iteration reaches the neighbors of $u_p$, we already have $u_p$ in $X$. This means that no matter what subset of $P$ is appended to $R$ after this point, $u_p$ will always stay in $X$, giving rise to a non-maximal cliques. Equipped with this knowledge, we can simply choose not to make the recursive calls which add neighbors of $u_p$ to $R$ at this point.

**Random and greedy choices of the pivot.** In [Koc01], two options are discussed to choose the pivot $u_p$. The first one consists of choosing the pivot at random. The second one consists of choosing as pivot the vertex with largest degree in $P$. In fact, for random graphs, this choice yields results much better than a random selection.

However, for a particular class of graphs [Koc01], the performance can be as poor or even poorer than a random selection. A single pivot determining the outcome for large number of disconnected vertices leads to many non-maximal cliques. As illustrated on Fig.4, consider any point in the algorithm where we have a large clique present in $P$, and a pivot which is disconnected from this clique. We can see that each node in this clique will be added to $R$ one after the other, regardless of whether it is part of any other clique. (Notice that the degree of nodes in the branches rooted at $1 \ldots n$ is one due to the pivot check.) See figure 4 for the recursion tree of $\Omega(n^2)$ size that will be formed in this case. (On Figs. 4 and 5, the dashed edges materialize the completion of the graph up to $n$ nodes. Also, the labels of the edges are the nodes that are added to $R$ when making that particular recursive call.)

Taking a cue from this construction, we can even construct cases where performance of the random **and** greedy policies decreases drastically. For example, if the situation in figure 5 exists at any point during the program, it is a straightforward verification that the size of the recursion subtree rooted at that point is $\Omega(n\sqrt{n})$.

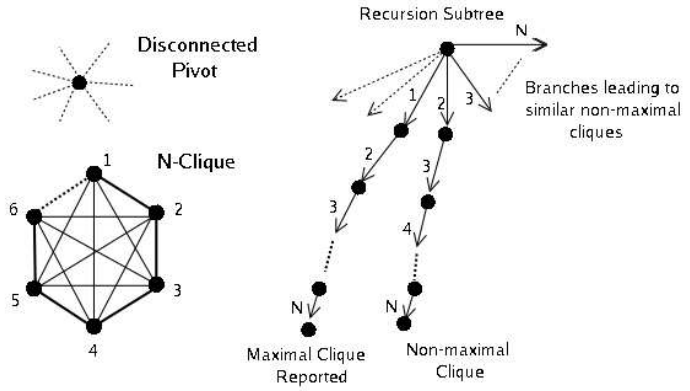


Figure 4: A Pivot disconnected from a clique yields a $\Omega(n^2)$ recursion tree —with $N = n$.

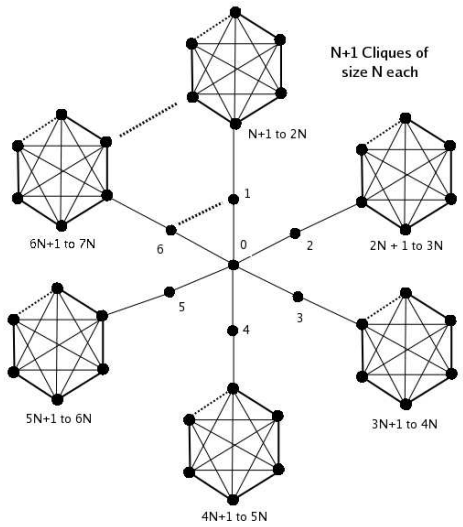


Figure 5: Random and greedy pivot selection yield a $\Omega(n\sqrt{n})$ size recursion tree, where $n$ is the total number of nodes —with $n = N^2$.

# 3 A pivot selection strategy eradicating worst-cases

As observed in [Koc01], the pivot can also be selected from $X$, since it maintains the same connectivity to $R$ as $P$. However, the meaning of pivot selection from $X$ is not clear from [Koc01], as the set $X$ can be empty. Moreover, no advantage/disadvantage of selecting pivot from $X$ is reported.

As depicted in fig. 4, if pivot is selected only from $P$, at every recursive call with non-empty $P$, the main loop in the algorithm is run at least once (exactly once in the situation in figure 4 ) since the pivot is a non-neighbor of itself. Being able to select the pivot from $X$ removes this anomaly, since all the vertices in $P$ can be skipped if they are neighbors to the pivot —Fig. 6. More precisely:

**Observation. 1** *A greedy strategy of pivot selection from the set $P \cup X$ does not suffer from the worst cases in fig. 4 and 5. The size of the recursion tree formed in such a case is $O(n)$.*

Interestingly, this pivot strategy not only eradicates worst-cases, but also improves the overall performances of the algorithm: whenever a situation similar to the worst-case occurs —a pivot with large degree disconnected from a clique, the quadratic trap is avoided. See section 5 for experimental results.

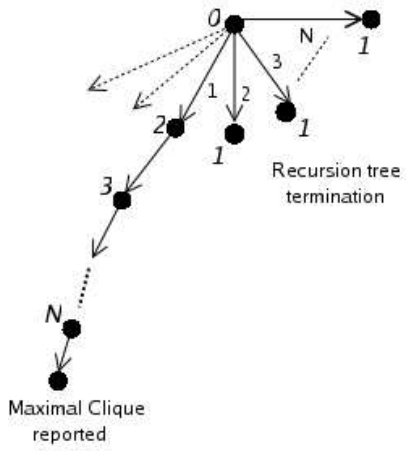


Figure 6: Elimination of the worst case by pivot selection from $P \cup X$. Compare with Fig. 4.

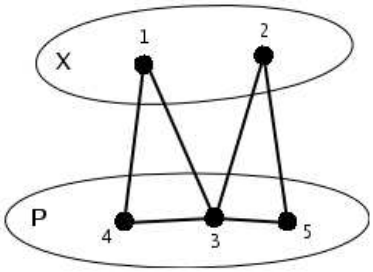


Figure 7: An example showing the necessity of continued checks for dominated nodes

# 4 Towards output sensitive algorithms

In this section, we investigate the recursive structure of cliques, and derive output sensitive algorithms. We introduce the concept of *dominated* nodes, and we show that I. Koch's pivot strategy is, in a sense, covered by this concept.

## 4.1 Removing redundant nodes

Our goal is to identify recursive calls which do not yield maximal cliques, and more precisely the nodes these calls are anchored at. To this end, we make the following:

**Definition. 1** *A node is said to be* redundant *at a particular recursion point, if there can be no maximal cliques formed which include the node, from that point onwards. A recursive call anchored at a redundant node is termed of* useless *or* non-productive *call.*

It is clear that a redundant node appended to $R$ at any point would lead to lot of unnecessary operations, as by definition, we will not reach any maximal clique in that recursion subtree. Hence it is imperative to identify and remove redundant nodes from $P$. We observe that nodes meeting certain condition, which we call *dominated* nodes are redundant, and hence can be removed from $P$ leading to increased efficiency.

**Strongly dominated nodes.** On the way to identify redundant nodes, we define:

**Definition. 2** *A node u is said to* dominate *node v with respect to a set S if and only if*

1. *Edge* $(u,v) \in E[G]$
2. $\forall w \in S,\ (v,w) \in E[G] \Rightarrow (u,w) \in E[G]$

*Note that by definition a node cannot have an edge to itself.*

**Observation. 2** *If at any recursion point during the execution of BK algorithm, a node* $v \in P$ *is dominated by a node* $u \in X$ *w.r.t. the set P, then v is redundant.*

*Proof.* Consider any path to a leaf node in the recursion tree, starting from the current recursion point. Let $U = \{u_0, u_1, ..., u_k\}$ be the set of nodes added to R along this path, not necessarily in the same order. Let $v = u_0$, and assume $v$ is dominated by $u \in X$.

Trivially, $U \subseteq P$. Also, since $R \cup U$ is a clique, $U$ itself is a clique. Therefore, $\forall\ 1 \leq i \leq k,\ (u_0,u_i) \in E[G]$. Since $u$ dominates $u_0$, $\forall\ 0 \leq i \leq k,\ (u,u_i) \in E[G]$. Also, since $u \in X$, $u$ is a neighbor to all nodes in $R$.

We have proved that $u$ is a neighbor to all nodes in $R \cup U$. So the clique $R \cup U$ can be extended by adding $u$ and hence, is non-maximal. □

Because of the above, we can now remove dominated vertices from $P$. The validity of this operation can be justified by the following : Consider the dominated node $v$ is the next to be 'selected' and added to $R$. After the corresponding recursive call returns, we add it to $X$ and continue. However,

because of the above proof, we know that the recursive call will not report any new clique. Hence, we just omit the recursive call. Further, we can omit adding $v$ to $X$ because node $u$ which dominates $v$ is such that:

1. From current recursion point onwards, any clique which can be extended by $v$ can also be extended by $u$. This ensures that no non-maximal cliques are reported, despite absence of $v$ from $X$.

2. Any node dominated by $v$ *w.r.t* any of the subsets of $P$ is also dominated by $u$, hence $v$ is not needed for removal of any dominated node.

The above properties are easily derived from definition 2.

Another issue here is that once a node is removed from $P$, the neighbor relations in $P$ change so that new dominated nodes arise. Hence the removal of dominated nodes should be repeated until no node in $P$ is dominated. For example, consider the situation in Fig. 7. Since node 1 and 2 are already in $X$, both the cliques $\{1,3,4\}$ and $\{2,3,5\}$ containing node 3 have been reported. Hence node 3 is redundant, but clearly it is undominated. After node 4 or node 5 are removed (since they are dominated by 1 and 2 respectively), we can see that node 3 will however be dominated, and can be removed.

**Weakly dominated nodes.** As just observed, being dominated w.r.t. $P$ is a strong condition for a node to be redundant. If all neighbors of $v \in P$ do not form a clique but only subsets of these neighbors do, we can actually restrict the dominance check to each such subset. This yields the following weaker condition:

**Observation. 3** *A node $v \in P$ which is dominated by some node in $X$ w.r.t. each of the connected components of the induced graph consisting of all its neighbors in $P$, is redundant.*

*Proof.* Consider any path to a leaf node in the recursion tree, starting from the current recursion point and adding $v$ to $R$ in the first step. Let $U = \{u_0, ..., u_k\}$ be the set of nodes added to R along this path after $v$, not necessarily in the same order.

Since $R \cup U$ is a clique, $U$ is also a clique. Also, the algorithm requires $U \subseteq (N[v] \cap P)$. Therefore, all the nodes in $U$ must belong to the same connected component of the induced graph of the neighbors of $v$ in $P$. But $\exists u \in X$ such that $u$ is neighbor to all neighbors of $v$ in $P$. Hence $u$ shall remain in $X$ at the end, making $R \cup U$ a non-maximal clique.

We have proved that any clique from current recursion point onwards, which contains $v$ must be non-maximal. Hence $v$ is redundant. □

**Ultimate definition of dominance.** Weakly dominated nodes are redundant, but the converse does not hold. Indeed, given a connected component containing a clique, the connected component may not be dominated while the clique is so. In other words, a sufficient and necessary dominance definition capturing all redundant nodes is the following:

**Definition. 3** *A node $v \in P$ which is dominated by some node in $X$ w.r.t. each subset $S$ of all its neighbors in $P$, where $S$ is a clique, is redundant.*

It is easy to see that this definition captures all redundant nodes. In fact, it is just a translation of the definition of a redundant node itself. Quite clearly, $X$ will be non-empty when any subset containing $v$ is added to $R$, and hence all the cliques formed containing $v$ shall be non-maximal.

However, the above definition, 'all subsets of neighbors forming a clique', induces an exponential cost (in the number of neighbors), and is this of little practical use. Practically, we shall restrict to the strong and weak definitions, which can be checked in polynomial time.

## 4.2 The pivot strategy anticipates the identification of dominated nodes

To check that I. Koch's pivot strategy is a particular case of the dominance concept, recall the decomposition of $P$ as $P = P^+ \cup P^-$ introduced in section 2.2. Once $u_p$ has been processed and added to $X$ —for the case $u_p \in P$, all nodes from $P^-$ are dominated by $u_p$. These nodes can be removed from $P$, an effect same as that of pivot selection. Summarizing, we have:

- The pivot strategy is a look-forward heuristic which anticipates the identification of dominated nodes *before* the corresponding recursive calls are made. Only the neighbors $P^-$ of the pivot in $u_p$ can be identified.
- The dominance check is a look-backward heuristic which uses information gained so far. The check is systematic in the sense that all nodes from $P$ are checked for dominance by nodes of $X$.

From this discussion, it is clear that some redundant nodes which are processed by the pivot strategy can be identified and discarded by dominance checks :

**Observation. 4** *During an execution of IK_*, nodes of $P^+$ which are dominated by a node of $X$ can be identified and discarded by the dominance check.*

## 4.3 Modified Bron-Kerbosch algorithms

**Modified BK algorithm.** In designing a modified BK algorithm, several options deserve discussion.

1. First, one needs to adopt one definition of dominance —which determines the cost of identifying such nodes. In this section, we develop algorithms templated by a definition of dominance. In section 5, we provide results for the strong and weak dominance definitions.

2. A second issue is to decide the exact point at which one should check for redundant vertices. One can see that at the beginning of a new iteration of the main loop, if a redundant vertex is in $P$, there is every possibility that it will be selected and added to $R$, giving rise to non-maximal cliques. Hence, it is obvious that the checks should be made just before this point.

The pseudocode for the modified Bron-Kerbosch algorithm and for the `removeDominated` function are displayed on Fig. 8, where the _* suffix indicates the type of dominance used is not specified. To recapitulate in brief, the optimizations stem from two facts:

1. Redundant nodes are removed cutting down a whole recursion subtree.
2. Dominated nodes are no more added to $X$, which speeds up the dominance checks further due to smaller size of $X$.

Once dominated nodes have been removed, a point to note here is that the issue of order in which nodes to select for examination from $P$ is still unresolved. One thing is certain, selecting a vertex from $P$ which is dominated by another from $P$ w.r.t. $P$ offers no benefit, because it is easy to work out that this will lead to non-maximal cliques. Hence choosing a vertex which is not dominated by any other node from $P$ would be a good choice. But beyond this partial order, establishing a total order seems difficult —recall Def. 3.

**Mixing the pivot strategy and dominance checks.** In section 4.2, we have seen that the nodes adjacent to the pivot are recognized as dominated automatically, without any dominance check. We would rather then classify them as dominated directly from the pivot strategy —avoiding costly dominance checks. The two strategies (pivot + dominance checks) can be used jointly, and algorithms using both are expected to outperform algorithms using a single strategy. An algorithm mixing the two strategies, templating the definition of dominance and choice of pivot —whence the suffix _*, is found in Fig. 9.

**A compromise.** During the run of an algorithm which uses mixed strategies as above, we can see that new recursive calls are introduced only by members of $P^+$, non-neighbors of the pivot, since all the neighbors are skipped. It thus makes sense to restrict the dominance checks to $P^+$. Especially in higher density graphs, size of $P^+$ tends to be small and this modification can brings about significant improvement in the running time. Hence we have:

**Observation. 5** *Attempting to remove only those dominated nodes not adjacent to the pivot improves the complexity of dominance checks.*

Notice that we do not make any statement regarding the overall complexity of the algorithm, since the gain on the recursion tree size may be eclipsed by the costs of the dominance checks. As we shall see later, the algorithm corresponding to this combination (called MBK_SDP$^+$_GPX in the next section), strikes a balance between costly dominance checks, and number of recursive calls made. In other words, this algorithm represents a compromise between performance on theoretical measures and practical measures.

Algorithm call: MBK_$*(\emptyset, V[G], \emptyset)$.

MBK_$*(R, P, X)$

```
1: if P = φ and X = φ then
2:    ReportClique(R)
3: else
4:    repeat
5:       removeDominated(P, X)
6:       if P = φ then
7:          break
8:       Let u_i ∈ P
9:       P ← P − {u_i}
10:      R_new ← R ∪ {u_i}
11:      P_new ← P ∩ N[u_i]
12:      X_new ← X ∩ N[u_i]
13:      MBK_*(R_new, P_new, X_new)
14:      X ← X + {u_i}
15:   until P ≠ φ
```

REMOVEDOMINATED$(P, X)$

```
1: while ∃v ∈ P, dominated by u ∈ X do
2:    Remove v from P
```

Figure 8: Modified Bron-Kerbosch Algorithm

Algorithm call: MBK_$*(\emptyset, V[G], \emptyset)$.

MBK_$*(R, P, X)$

```
1: if P = φ and X = φ then
2:    ReportClique(R)
3: else
4:    Let u_p be the pivot selected from P ∪ X.
5:    repeat
6:       removeDominated(P, X)
7:       Let N_p be the set of non-neighbors of u_p in P.
8:       if N_p = φ then
9:          break
10:      Let u_i ∈ N_p
11:      P ← P − {u_i}
12:      R_new ← R ∪ {u_i}
13:      P_new ← P ∩ N[u_i]
14:      X_new ← X ∩ N[u_i]
15:      MBK_*(R_new, P_new, X_new)
16:      X ← X + {u_i}
17:   until P ≠ φ
```

REMOVEDOMINATED$(P, X)$

```
1: while ∃v ∈ P, dominated by u ∈ X do
2:    Remove v from P
```

Figure 9: Modified Bron-Kerbosch Algorithm with pivot selection incorporated

## 4.4 Complexity and Memory requirement

So far, we assumed one knew how to greedily select a pivot, or remove dominated nodes. Although the complexity of these operations depends on the particular way sets are represented, let us analyze their cost assuming sets are represented using dictionaries [1]. For convenience, denote $d(\mid S \mid)$ the cost of a dictionary operation on a set $S$.

Two operations deserve attention in this context : The greedy choice of pivot and the dominance checks. Both these operations involve for every node $v \in S$, $S$ being either $P$ or $P \cup X$, the set of neighbors of $v$ that lies in $P$. Since our input consists of neighborhood relations for the entire graph, this can be done in $\mid N[v] \mid = O(\mid V[G] \mid) = O(n)$ dictionary operations on set $P$. The same can also be achieved by iterating over $P$ and making $\mid P \mid$ dictionary operations on $N[v]$. Since greedy choice of pivot just uses the cardinality of these calculated sets, its complexity is $O(n \mid S \mid d(\mid P \mid))$

[1] A dictionary operation with a balanced binary search tree (perfect hash table) takes worst-case time $O(\log n)$ (amortized $O(1)$ time).

or $O(|S||P|d(|N[v]|))$ according to our choice. In practice, we can determine the larger of the two sets $P$ and $N[v]$ and choose to perform the dictionary operations on it.

Consider now dominance checks corresponding to Def. 2. Given a node $v$ in $P$ and $u$ in $X$, we can determine if $u$ dominates $v$, by simply iterating over the above calculated neighborhood of $v$ and checking if $u$ is a neighbor to all its members. Since the size of above set as well as $N[u]$ is $O(n)$, we can determine complexity of the `removeDominated` function call to be $O(n|P||X|d(n))$.

Notice also that the pivot is selected once, while the `removeDominated` function is called iteratively.

Turning to the memory requirements, we observe that $\Omega(n^2)$ memory is inevitably required for any of the mentioned algorithms, since we can make recursive calls up to depth $n$, each requiring $O(n)$ memory just for storing the sets $R$, $P$ and $X$. (Notice the quadratic bound is achieved for a complete graph.) In practice, we can use more memory space to speed the execution up. The algorithm frequently uses set of neighbors of a given node in $P$ or $X$. Since our input consists of neighborhood relations over the whole graph, an intersection operation is required every time we deal with the above sets. Instead, we can precompute and store this information, to avoid these operations. This obviously requires $|P|^2$ or $|P||X|$ space per recursive call, increasing the overall requirement to $O(n^3)$.

# 5 Experimental Results

This section reports experiments on the standard BK algorithm, two variants by I. Koch, and five alternatives stemming from the insights developed in this paper. In particular, we investigate the trade-off between output sensitivity and the overhead associated to the identification of dominated nodes.

## 5.1 Algorithms and implementations issues

**Algorithms.** We shall compare three groups of algorithms. First, the standard BK together with the variants of I. Koch:

1. **BK** : The original Bron-Kerbosch algorithm.
2. **IK_RP** : BK algorithm with heuristic in [Koc01] using random pivot selection from $P$. Template code on Fig. 3.
3. **IK_GP** : BK algorithm with heuristic in [Koc01] using greedy pivot selection policy from $P$. Template code on Fig. 3.

Next, to get rid of worst-cases, we introduce:

1. **IK_GPX** : BK algorithm with heuristic in [Koc01] using greedy pivot selection from $P \cup X$. Template code on Fig. 3.

Finally, we develop four Modified BK algorithms:

1. **MBK_SD** : Modified BK algorithm with removal of strongly dominated nodes —refer to observation 2. Code on Fig. 8.
2. **MBK_SD_GPX** : Modified BK algorithm with removal of strongly dominated nodes as well as a greedy pivot selection policy from $P \cup X$. Template code on Fig. 9.
3. **MBK_SDP$^+$_GPX** : The variant of MBK_SD_GPX which restricts removal of strongly dominated nodes to non-neighbors $P^+$ of the pivot. Refer observation 5. Template code on Fig. 9.
4. **MBK_WD_GPX** : Modified BK algorithm with removal of weakly dominated nodes — observation 3, as well as a greedy pivot selection policy from $P \cup X$. Template code on Fig. 9.

Finally, one last comment. For the four MBK algorithms, we resolve the issue of selection order of nodes in $P$ by iteratively selecting the node having largest degree in $P$. As remarked in section 2.1, a prudent choice would be to select a node not dominated by any other in $P$ *w.r.t.* $P$. It is obvious that the largest degree node cannot be dominated —by breaking ties in its favor if more than one largest degree nodes exist. Note here that this selection order is unrelated to the greedy pivot selection policy. Pivot selection happens only once in a recursive call, whereas the above is performed in every iteration of the primary loop in the algorithm.

**Data structures and implementation choices.** All above algorithms manipulate subsets of the vertices $V[G]$ of the graph $G$. Representing any subset of $V[G]$ as a boolean string of length $\mid V[G] \mid$ has the advantage of providing constant time query and update operations. But such a representation requires worst-case storage, a drawback upon recursive calls. Another constraint comes from the dictionary operations required by dominance checks. For these reasons, sets are most naturally represented as dictionaries, either hash tables or balanced binary search trees.

In order for all implementations to be comparable, we implemented all algorithms using the Java 1.4.2 Collection framework. More precisely:
—Sets are stored in hash tables —LinkedHashSets from Java 1.4.2 Collection framework. This data structure guarantees amortized $O(1)$ insertion, removal and query time. All the elements in the set are also connected as a linked list, which allows traversal in linear time.
—The checks for detecting dominated nodes have been implemented using a candidate queue. We can observe that the only nodes which can 'become' dominated after a dominated node is removed from $P$ are its neighbors. Hence, we pop off nodes successively from the queue to check for dominance, and if found dominated, we add its neighbors to the queue.

## 5.2 Experimental Conditions

**Random graphs.** Our experimental study focuses on random graphs since these are easily reproducible —hence allowing comparisons with the study of I. Koch, and also span a wide range of

structural properties. The specification of a random graph involves the number $n$ of nodes, and the probability $p$ of an edge being present. Naturally, all values reported correspond to *averages* over several random graphs.

**$n$ versus $p$.** We first observe experiments of I. Koch are for low density graphs only —$p \leq 0.3$. We provide results for $p \in [0,1]$ on graphs of various sizes. Exploring the whole range of $p$ is actually fundamental since experimental evidence shows the problem of reporting all maximal cliques is much harder for values of $p$ near 0.9 —Fig. 11.

**Machine.** All test were run on a Pentium at 3Ghz with 2.5 Go of RAM. All programs were implemented with the Java 1.4.2 Collection framework.

**Admissible ranges.** For applications, it helps to know for which kind of graphs the problem can be solved in a reasonable amount of time. As we shall see, the reference algorithm is IK_GPX. Setting an upper bound of say one minute, table 1 gives for a given edge probability $p$ the maximum value of $n$ such that an average execution lasts less than 1 minute.

## 5.3 Performance Criteria

The total running time of the algorithm stems from the number of recursive calls and the polynomial time operations associated to a call in the body of the recursive function. To account on these sources, the following parameters are of interest: the number #r.c. of recursive calls made; the number #u.r.c. of non-productive calls; the system time $t$ in seconds.

## 5.4 Comparative analysis

In this section, all values reported corresponds to averages over 5 runs of the algorithms on input graphs with prescribed values of $n$ and $p$. The average values of #cliques, #r.c. and #u.r.c. have been rounded down to integers.

**Sample values.** Tables 2 and 3 list the performance statistics for two sample values of the parameters. As shown on Fig. 12 —which features the four top algorithms, the results for other values of of $n$ and $p$ show similar gradation. One can immediately see, that all the heuristics bring about a large improvement in the performance of BK algorithm, by all measures and that the original BK algorithm is forbiddingly slow for $p = 0.7$ This contrasts with the observation raised in [Koc01], where all algorithms examined, including BK, having running times within a factor three. But as mentioned above, high values of $p$ yield more difficult problems.

**Non-productive calls.** The gradation in the number of non-productive calls made by various algorithms is the most noticeable. All the algorithms which use removal of redundant nodes, MBK_SD, MBK_SD_GPX and MBK_WD_GPX, outperform their contenders in this measure. This gradation is in line with the strictness of the condition of redundancy used by various algorithms : IK_RP, IK_GP and IK_GPX, which use the strongest condition, perform poorly whereas MBK_SD and MBK_SD_GPX which use a much weaker condition, perform a lot better. MBK_SDP$^+$_GPX,

which is a compromise between MBK_SD_GPX and IK_GPX, has intermediate performance. Finally, MBK_WD_GPX, which uses the weakest condition, makes the least number of non-productive recursive calls. To conclude, removal of redundant nodes brings about a large improvement in the theoretical output sensitivity of the BK algorithm.

**Running time.** For the running time of the algorithms, we shall discuss comparatively the fastest variants of our algorithms, IK_GPX, MBK_SD_GPX, MBK_SDP$^+$_GPX and their fastest alternative, IK_GP. First of all, IK_GPX outperforms IK_GP in all measures. The reason for this improvement may be removal of the worst case, as explained in section 3. A situation similar to the one in fig. 4 can arise multiple times during execution of the algorithm, and the advantage gained by IK_GPX in such situations accumulates to yield the overall improvement.

Turning to MBK_SD_GPX and MBK_SDP$^+$_GPX, the values of system time required for these variants show that with the current implementation, the advantage gained by large reduction in the size of the recursion tree has not appeared as improvement in running time with the same magnitude. Quite clearly, the overheads of dominance checks add a significant amount to the running time. This is especially true in smaller and less dense graphs. For such graphs, the size of recursion tree itself is small, and hence not a lot of advantage can be gained by removing redundant nodes. The cost of dominance checks thus becomes significant compared to overall time.

For large and dense graphs however, the benefit imparted by removal of a redundant node is hefty, and the resulting improvements can be seen in table 4. In case of MBK_SD_GPX, the improvement is smaller : Despite making less than half recursive calls as IK_GP, the running time improves only by a factor of approximately 1.5, the reason being large overheads incurred due to dominance checks. IK_GPX, which works on the efficient heuristic of pivot selection, brings about a large reduction in running time.

On this background, we can see how MBK_SDP$^+$_GPX is a good compromise between MBK_SD_GPX and IK_GPX. Requiring almost same amount of system time as IK_GPX, it reduces the number of non-productive calls significantly. Notice however the number of useless calls of MBK_SDP$^+$_GPX is larger than that of MBK_SD_GPX. To see why, consider a node $v \in P^+$. During the execution of MBK_SDP$^+$_GPX, nodes from $P^-$ remain in the neighborhood of $v$. Given the iterative structure of the removal of dominated nodes, this prevents the identification of nodes within $P^+$ which would have been dominated upon removal of some nodes from $P^-$.

**Worst cases.** Apart from this, removal of redundant nodes gets rid of the worst case faced by IK_GP. The results for input graphs of various sizes derived from the idea in figure 4 can be found in tables 5, 6 and 7. MBK_SD_GPX removes this asymmetry by running the dominance checks on all nodes. MBK_SD_GPX outperforms IK_GP in all measures including running time, when run on this kind of graphs.

As explained in section 2.2, IK_GPX also gets rid of the worst case. However rather than introduction of symmetry, this fix works by providing a convenient pivot, hence is in some sense, ad-hoc.

# 6 Further attempt at exploiting the structure of cliques

We have seen that pivot selection offers very efficient means of filtering out redundant nodes, but it is less effective in filtering out redundant nodes than dominance checks. Hence, an obvious consequent is to somehow extend the pivot selection strategy. The deficiency in pivot selection is that redundant nodes in $P^+$ are processed in the main loop without being monitored —which also accounts for the fact MBK_SDP$^+$_GPX performs better at eliminating non-productive calls.

Our aim would be not to process all the nodes in $P^+$ categorically, but to somehow defer their processing until we figure out redundancies in $P^+$. As mentioned above, we would like the means of figuring out these redundancies to be pivot selection, since it is very efficient. In short, we would like to select multiple pivots, moreover, these pivots must be from $P^+$.

To achieve this we split the set $P$ in the function call BK$(R,P,X)$ into $P$ and $Q$. Set $P$ holds the set of nodes from which pivot is to be selected, i.e. the nodes which are to be processed in the main loop potentially, whereas $Q$ holds the nodes that are not to be processed in the main loop of current recursive call. The invariant is that all the maximal cliques to be discovered in the current recursion subtree consist of at least one node from $P$ and zero or more from $Q$. Keeping this in mind, we observe that all the maximal cliques to be found in the current recursion subtree can be partitioned into following three types :

1. Cliques containing the pivot and zero or more of nodes from $P^- \cup (Q \cap N[u_p])$.
2. Cliques containing one or more nodes from $P^+ - \{u_p\}$ and zero or more from $P^- \cup Q$.
3. Cliques containing one or more nodes from $P^-$ and zero or more from $Q$.

Translation of the above partition into recursive calls yields following algorithm :

Algorithm call: MBK_E$*(\emptyset, V[G], \emptyset, \emptyset)$.

MBK_E$*(R, P, Q, X)$

```
1: if P = φ then
2:    if X = φ then
3:       ReportClique(R)
4: else
5:    Let u_p be the pivot.
6:    MBK_E*(R ∪ {u_p}, P⁻ ∪ (Q ∩ N[u_p]), ∅, X ∩ N[u_p])
7:    MBK_E*(R, P⁺ − {u_p}, P⁻ ∪ Q, X ∪ {u_p})
8:    MBK_E*(R, P⁻, Q, X ∪ {u_p} ∪ P⁺)
```

Figure 10: Template algorithm for extended pivot selection

We can assign following meanings to the each of the three recursive calls in the function body :

1. Expansion : In the first recursive call, the current clique is expanded using the pivot.

2. Lookahead : The second recursive call essentially allows us to select more pivots before members of $P^+$ are added to the growing clique.

3. Recovery : However, in the first two recursive calls, we miss out on cliques which contain only nodes from $P^-$ and $Q$. Hence, we 'recover' them with the third call.

Note that the main loop found in all the previous algorithms has been replaced by recursive calls. In practice, this alternative does not compete with the top four algorithms of the previous section due to the overheads induced by the recursive calls.

# 7 Conclusion

By studying the structure of the *Maximal Cliques Detection* problem, this paper makes two contributions. First, we provide a modification of the usual greedy pivoting strategy. The corresponding algorithm, IK_GPX, eradicates worst cases of existing algorithms, and can be implemented efficiently. However, there seems to be little possibility of any extensions to pivot selection.

Second, we introduce *Dominance*, a concept whose instantiation yields algorithms reducing significantly the number of recursive calls. While other variants incur a large penalty for dominance checks, algorithm MBK_SDP$^+$_GPX strikes a balance between dominance checks and number of recursive calls required. Further work on dominance checks and their complexity shall lead to faster algorithms.

**Acknowledgment.** C. Karande acknowledges the support of the INRIA internship program.

# References

[BBPP99] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization, Vol. 4*. Kluwer, 1999.

[BK73] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Comm. ACM*, 16(9):575–577, 1973.

[CK05a] P. Agrawal F. Cazals and C. Karande. Partial combinatorial shape matching through product graphs. 2005.

[CK05b] P. Agrawal F. Cazals and C. Karande. Partial geometric shape matching through product graphs. 2005.

[GARW93] H.M. Grindley, P.J. Artymiuk, D.W. Rice, and P. Willett. Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm. *J. Mol. Biol.*, 229, 1993.

[GWA00] E.J. Gardiner, P. Willett, and P.J. Artymiuk. Graph-theoretic techniques for macromolecular docking. *J. of Chem. Inf. and Comp. Sc.*, 40, 2000.

[Koc01] I. Koch. Fundamental study: Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Comp. Sc.*, 250(1-2):1–30, 2001.

[KWF01] F. Kose, W. Weckwerth, T. Linke 2, and O. Fiehn. Visualizing plant metabolomic correlation networks using clique-metabolite matrices. *Bioinformatics*, 17(12), 2001.

[SBK92] B.K. Stoichet, D.L. Bodian, and I.D. Kuntz. Molecular docking using shape descriptors. *J. Comp. Chem.*, 13(3), 1992.

[SM98] R. Samudrala and J. Moult. A graph-theoretic algorithm for comparative modelling of protein structure. *J. of Mol. Bio.*, 279, 1998.

# 8 Appendix: experimental results

**Admissible ranges**

| $p$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|
| $n$ | 880 | 520 | 320 | 210 | 150 | 110 | 80 | 60 |

Table 1: Maximum size of the graph for which IK_GPX runs within one minute

**Difficulty as a function of $p$**

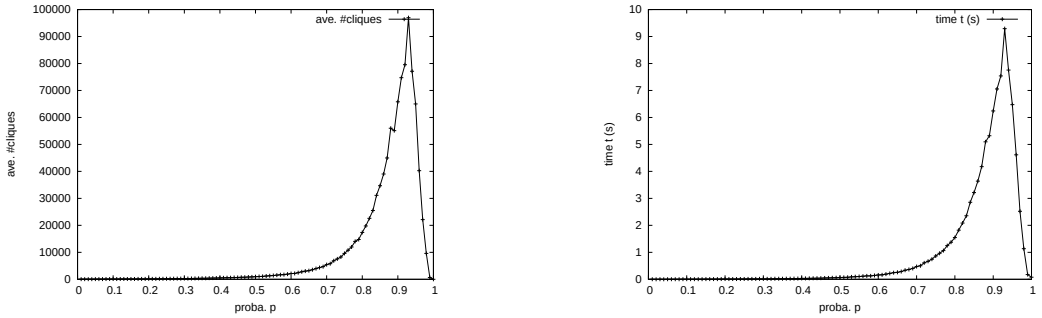


Figure 11: Averages over 50 runs for graphs containing 50 nodes (a)Number of cliques (b)Running time of IK_GPX

**Sample values**

| | #cliques | #r.c. | #u.r.c. | $t$ |
|---|---|---|---|---|
| BK | 16151 | 201228 | 153879 | 4.06 |
| IK_RP | 16151 | 92898 | 55053 | 2.32 |
| IK_GP | 16151 | 55059 | 18876 | 1.23 |
| IK_GPX | 16151 | 39076 | 3497 | 1.03 |
| MBK_SD | 16151 | 37024 | 676 | 2.02 |
| MBK_SD_GPX | 16151 | 34482 | 110 | 1.35 |
| MBK_SDP$^+$_GPX | 16151 | 35210 | 679 | 1.12 |
| MBK_WD_GPX | 16151 | 34373 | 49 | 2.07 |

Table 2: Results for input graphs containing 100 nodes with edge probability of 0.5

| | #cliques | #r.c. | #u.r.c. | *t* |
|---|---|---|---|---|
| BK | 80391 | 4857591 | 4535847 | 129.16 |
| IK_RP | 80391 | 822127 | 586101 | 22.93 |
| IK_GP | 80391 | 313275 | 117713 | 7.34 |
| IK_GPX | 80391 | 195620 | 8501 | 5.12 |
| MBK_SD | 80391 | 212068 | 3506 | 14.85 |
| MBK_SD_GPX | 80391 | 181947 | 100 | 6.56 |
| MBK_SDP$^+$_GPX | 80391 | 184228 | 1405 | 5.50 |
| MBK_WD_GPX | 80391 | 181856 | 71 | 10.33 |

Table 3: Results for input graphs containing 80 nodes with edge probability of 0.7

**Comparisons**

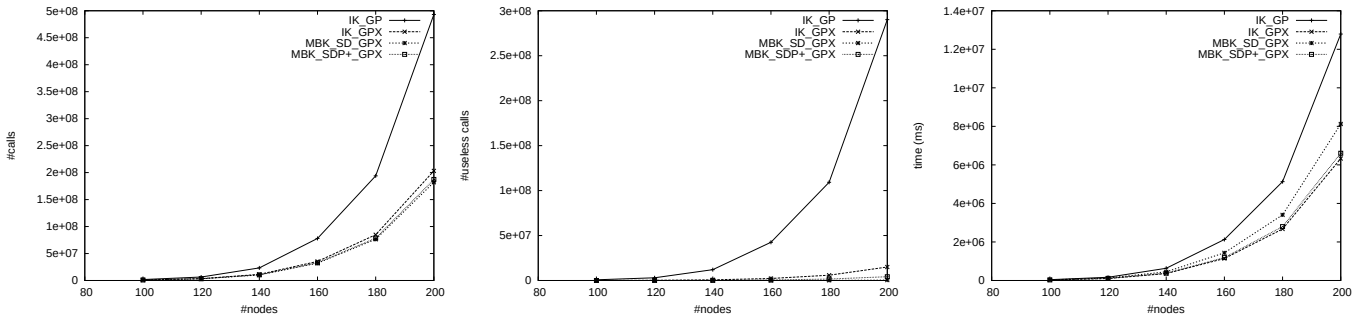


Figure 12: (a)No. calls (b)No. useless calls (c)System time for large graphs with edge probability of 0.7.

| | #cliques | #r.c. | #u.r.c. | *t* |
|---|---|---|---|---|
| IK_GP | 75065882 | 493163638 | 290139107 | 12791.28 |
| IK_GPX | 75065882 | 203149445 | 14976619 | 6329.27 |
| MBK_SD_GPX | 75065882 | 182024997 | 389072 | 8123.37 |
| MBK_SDP$^+$_GPX | 75065882 | 187380234 | 4196343 | 6605.29 |

Table 4: Results for input graphs containing 200 nodes with edge probability of 0.7.

**Worst cases**

| No. of nodes | IK_GP | IK_GPX | MBK_SD_GPX | MBK_SDP$^+$_GPX |
|---|---|---|---|---|
| 42 | 234 | 63 | 44 | 44 |
| 82 | 864 | 123 | 84 | 84 |
| 122 | 1894 | 183 | 124 | 124 |
| 162 | 3324 | 243 | 164 | 164 |
| 202 | 5154 | 303 | 204 | 204 |

Table 5: No. of recursive calls for worst case graphs

| No. of nodes | IK_GP | IK_GPX | MBK_SD_GPX | MBK_SDP$^+$_GPX |
|---|---|---|---|---|
| 42 | 189 | 18 | 0 | 0 |
| 82 | 779 | 38 | 0 | 0 |
| 122 | 1769 | 58 | 0 | 0 |
| 162 | 3154 | 78 | 0 | 0 |
| 202 | 4949 | 98 | 0 | 0 |

Table 6: No. of non-productive calls for worst case graphs

| No. of nodes | IK_GP | IK_GPX | MBK_SD_GPX | MBK_SDP$^+$_GPX |
|---|---|---|---|---|
| 42 | 0.08 | 0.05 | 0.06 | 0.05 |
| 82 | 1.00 | 0.15 | 0.23 | 0.08 |
| 122 | 4.32 | 0.33 | 0.25 | 0.22 |
| 162 | 15.26 | 0.67 | 0.53 | 0.34 |
| 202 | 29.72 | 1.08 | 0.79 | 0.57 |

Table 7: Average running time for worst case graphs

# Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399