Honour Thy Neighbour—Clique Maintenance in Dynamic Graphs

Thorsten J. Ottosen
Department of Computer Science, Aalborg University, Denmark
nesotto@cs.aau.dk

Jiří Vomlel
Institute of Information Theory and Automation of the AS CR, The Czech Republic
vomlel@utia.cas.cz

Abstract

Whenever objects and their interaction is modelled via undirected graphs, it is often of great interest to know the cliques of the graph. For several problems the graph changes frequently over time, and we therefore seek methods for updating the information about the cliques in a dynamic fashion to avoid expensive recomputations. This dynamic problem was investigated by Stix, and in this paper we derive a new simple method based on the Bron-Kerbosch algorithm that compares favourably to Stix' approach. The new approach is generic in the sense that it can be used with other algorithms than just Bron-Kerbosch. The applications include fuzzy clustering and optimal triangulation of Bayesian networks.

1 Introduction

We consider the problem of maintaining the set of cliques of a dynamic undirected graph. The graph is dynamic in the sense that edges can be removed and added, but the set of vertices is invariant. When we add a new set of edges, we call the problem *incremental*, and when we remove a set of edges, we call the problem *decremental*. Finding all the cliques of a static graph is a hard problem: the clique decision problem is NP-complete (Karp, 1972) and listing all the cliques may require exponential time as there exists graphs with exponentially many cliques (Moon and Moser, 1965)—albeit it is solvable in polynomial time for many classes of graphs. However, in this work we shall consider the initial set of cliques for given (several well-known algorithms exists for this purpose).

Previous research has been motivated by fuzzy clustering (Stix, 2004), but we have another application in mind. Specifically, our motivation is to find optimal triangulations of Bayesian networks with respect to the total table size by using a best-first or depth-first search. This requires a lower bound on the total table size for which we may use the total table size of the current partially triangulated graph. In turn, this requires that we know the cliques of the current graph. A detailed description of the new best-first and depth-first approach to triangulation can be found in (Ottosen and Vomlel, 2010).

We shall use the following notation and definitions. $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is an *undirected graph* with *vertices* $\mathrm{V} = \mathcal{V}(\mathrm{G})$ and *edges* $\mathrm{E} = \mathcal{E}(\mathrm{G})$. For a set of edges F, $\mathcal{V}(\mathrm{F})$ is the set of vertices $\{\mathrm{u}, \mathrm{v} : \{\mathrm{u}, \mathrm{v}\} \in \mathrm{F}\}$. For $\mathrm{W} \subseteq \mathrm{V}$, $\mathrm{G}[\mathrm{W}]$ is the *subgraph induced by* W. Two vertices u and v are *connected* in G if there is an edge between them. A graph G is *complete* if all pairs of vertices $\{\mathrm{u}, \mathrm{v}\}$ ($\mathrm{u} \neq \mathrm{v}$) are connected in G. A set of vertices $\mathrm{W} \subseteq \mathrm{V}$ is *complete* in G if $\mathrm{G}[\mathrm{W}]$ is a complete graph. If W is complete and no complete set U exists such that $\mathrm{W} \subset \mathrm{U}$, then W is a *clique*. (Remark: note that any complete set is sometimes called a clique; then what we defined as a clique is called a maximal clique.) The *set of all cliques* of a graph is denoted $\mathcal{C}(\mathrm{G})$ and the set of all cliques that intersects with a set of vertices W is denoted $\mathcal{C}(\mathrm{W}, \mathrm{G})$. For a single vertex v we also allow the notation

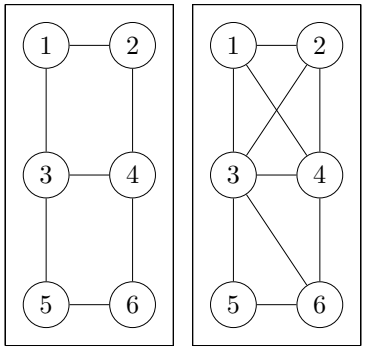

Figure 1: Left: The initial graph $G = (V, E)$. Right: The updated graph G'. We have $\mathcal{C}(G) = E$ and $\mathcal{C}(G') = \{\{1,2,3,4\},\{3,5,6\},\{3,4,6\}\}$. So in this example we have $\mathcal{RC}(G, G') = \mathcal{C}(G)$ and $\mathcal{NC}(G, G') = \mathcal{C}(G')$.

$\mathcal{C}(v, G)$. The *neighbours* of a set of vertices W are those vertices from $V \setminus W$ that are connected to at least one vertex $v \in W$ and we write this as $nb(W, G)$. Similarly, $fa(W, G)$ is the *family* of W in G, that is, the set $nb(W, G) \cup \{W\}$. As usual we allow the notation $nb(v, G)$ and $fa(v, G)$. A vertex v is *simplicial* if $nb(v, G)$ is complete. If $G' = (V, E \cup F)$ is the graph resulting from adding a set of new edges F to G, then $\mathcal{RC}(G, G') = \mathcal{C}(G) \setminus \mathcal{C}(G')$ is the set of *removed cliques*, and $\mathcal{NC}(G, G') = \mathcal{C}(G') \setminus \mathcal{C}(G)$ is the set of *new cliques*. Figure 1 illustrates these concepts. Finally, a complete set of vertices C in G' is called a *clique candidate* for G'.

2 Stix' Approach To Clique Maintenance

Stix observed that it was somewhat expensive to recompute all cliques of a graph given that the graph had only changed slightly. Therefore Stix derived the approach explained below and showed that it did indeed out-perform a full recomputation scheme (Stix, 2004).

Stix' approach works by adding (removing) one edge at a time. To add (remove) a set of edges, the technique is simply applied once for each edge. The technique (both for incremental and decremental problems) may be summarized as follows: (1) Let $G = (V, E)$ be an undirected graph, and let $G' = (V, E \cup \{\{v, w\}\})$. (2) Initially let $\mathcal{C} = \mathcal{C}(G)$. (3) Generate a set of clique candidates $\mathcal{K}$ for the updated graph G'. (4) Add/remove a candidate $C \in \mathcal{K}$ to/from $\mathcal{C}$ depending on whether it is a clique in G'. (5) In the end, $\mathcal{C}$ equals $\mathcal{C}(G')$

The check in step 4 is shown in Algorithm 1 where we have improved Stix' approach by only considering the neighbours $nb(C, G')$ of a clique candidate C (notice that this algorithm should be called with the updated graph G' as its second argument).

Stix' algorithm is based on the following theorem:

Theorem 1. *(Stix, 2004) Let* $G = (V, E)$ *be an undirected graph, and let* $G' = (V, E \cup \{\{v, w\}\})$ *be the graph after adding the edge* $\{v, w\}$. *Then*

1. *All cliques of* $\mathcal{C}(G)$ *that do not contain* v *or* w *are in* $\mathcal{C}(G')$.
2. *For all* $A \in \mathcal{C}(v, G)$ *and for all* $B \in \mathcal{C}(w, G)$ *we have*
   (a) $L = A \cap B \cup \{v, w\}$ *is a clique candidate for* G'.
   (b) $|A \setminus B| = 1 \implies A \notin \mathcal{C}(G')$*; otherwise* A *is a clique candidate for* G'.
   (c) $|B \setminus A| = 1 \implies B \notin \mathcal{C}(G')$*; otherwise* B *is a clique candidate for* G'.
3. *The set* $\mathcal{C}(G')$ *is fully determined by statement (1) and by inspecting all the clique candidates of statement (2).*

Stix' algorithm with several improvements is presented as Algorithm 2. Notice that the first part of condition 2(b) and 2(c) is not checked in Algorithm 2. We conducted experiments with these conditions being checked, but found it to be about 40% slower. Furthermore, we accumulate clique candidates in a set to reduce the number of calls to IsClique($\cdot$).

Next we illustrate how Stix' algorithm work in a small example.

Example 1. Consider the graph in Figure 2 on the left which we want to update with the set of edges $\{\{3,4\},\{3,5\}\}$. When adding the edge $\{3,4\}$, line 7-13 in Stix' algorithm combines the two sets of cliques $\mathcal{C}(3, G) = \{\{1,3\},\{3,6\}\}$

Algorithm 1 Verifying a complete set C is a clique (improved version)

```
1: function IsClique(C, G)
2:     Input: A non-empty, complete set of
3:             vertices C, and a graph G.
4:     for all v ∈ nb(C, G) do
5:         if C ⊆ nb(v, G) then
6:             return false
7:         end if
8:     end for
9:     return true
10: end function
```

and $\mathcal{C}(4, \mathrm{G}) = \{\{2,4,5\},\{4,5,6\}\}$. The resulting set of clique candidates is then $\mathcal{K} = \{\{3,4\},\{3,4\},\{3,4\},\{3,4,6\}\}$. Then follows a series of calls to `IsClique(·)` which determines that the clique $\{3,6\}$ must be removed from and the clique $\{3,4,6\}$ must be added to $\mathcal{C}$.

In the second iteration we add the edge $\{3,5\}$ and get the set of candidates $\mathcal{K} = \{\{3,5\},\{3,5\},\{3,4,5\},\{3,4,5,6\}\}$ and determine that the cliques $\{3,4,6\}$ and $\{4,5,6\}$ must be removed and the clique $\{3,4,5,6\}$ must be added.

The above example shows that there are two potential performance problems with Stix' approach when adding multiple edges:

1. Many duplicate clique candidates are generated and existing cliques are combined multiple times, and
2. A great number of calls to `IsClique(·)` is needed to prune candidates and remove old cliques.

To overcome these problems, one might try to generalize Stix' theoretical results to account for a larger set of edges being added at one time. However, it turns out that such an approach suffers even more from the problems above. In the following we shall therefore present a radically different approach that overcomes both problems.

3 Clique Maintenance by Local Search

The general idea behind this method is simple: instead of running the Bron-Kerbosch algorithm (or a similar algorithm) on the whole graph, run it on a smaller subgraph where all the new cliques appear and existing cliques disappear. Then simply update the set of cliques based on the vertices of the subgraph and the newly found cliques. Algorithm 3 is the modified Bron-Kerbosch algorithm which by using a pivot can reduce the search space (to get the original algorithm simply exchange the iteration in line 7 with "**for all** v ∈ P **do**"). In our implementation we pick the pivot deterministically as the first vertex in P because this is very easy (for alternative pivot selection strategies see (Cazals and Karande, 2008) and (Koch, 2001)).

To find all cliques of a graph G, the algorithm should be called with the argument tuple $(\mathrm{G}, \emptyset, \mathcal{V}(\mathrm{G}), \emptyset)$. However, an important ob-

Algorithm 2 Incremental clique maintenance by single-edge updates (improved version)

```
1: function EdgeBasedUpdate(C, G, F)
2:     Input: A graph G = (V, E),
3:             the set of cliques C of G, and
4:             the set of new edges F.
5:     for all {u, v} ∈ F do
6:         Let G' = (V, E(G) ∪ {{u, v}})
7:         Set K = ∅
8:         for all A ∈ C(u, G) do
9:             for all B ∈ C(v, G) do
10:                Let C = A ∩ B ∪ {u, v}
11:                Set K = K ∪ {C}
12:            end for
13:        end for
14:        for all K ∈ C(u, G) ∪ C(v, G) do
15:            if !IsClique(K, G') then
16:                Set C = C \ {K}
17:            end if
18:        end for
19:        for all K ∈ K do
20:            if IsClique(K, G') then
21:                Set C = C ∪ {K}
22:            end if
23:        end for
24:        Set G = G'
25:    end for
26:    return C
27: end function
```

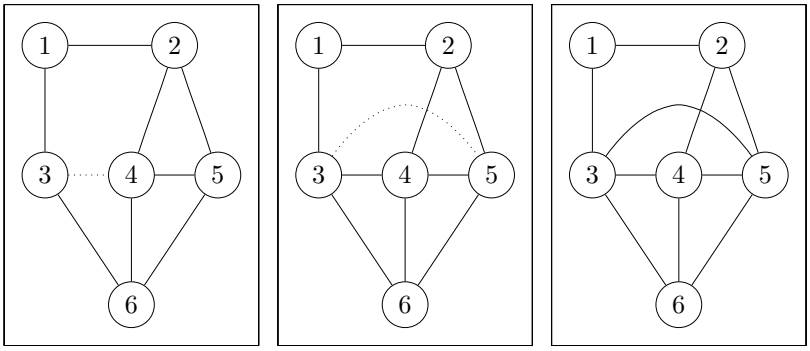

Figure 2: The sequence of graphs considered by Stix' algorithm (Algorithm 2) when adding the set of edges $\{\{3,4\},\{3.5\}\}$. Left: The initial graph—a dotted edge indicates it is about to be added to the graph. Middle: the graph after the first edge has been added. Right: The final graph.

Algorithm 3 The Bron-Kerbosch algorithm with pivot

```
1:  function BKWithPivot(G, R, P, X)
2:      if P = ∅ and X = ∅ then
3:          return {R}
4:      else
5:          Let C = ∅
6:          Let u = SelectPivot(P, X, G)
7:          for all v ∈ P \ nb(u, G) do
8:              Set P = P \ {v}
9:              Let R_new = R ∪ {v}
10:             Let P_new = P ∩ nb(v, G)
11:             Let X_new = X ∩ nb(v, G)
12:             Let K =
13:       BKWithPivot(G, R_new, P_new, X_new)
14:             Set X = X ∪ {v}
15:             Set C = C ∪ K
16:         end for
17:         Return C
18:     end if
19: end function
```

servation is that the algorithm can also search a subgraph G[W] for cliques by simply passing the arguments $(G, \emptyset, W, \emptyset)$. It is this ability that our new clique maintenance algorithm takes advantage of. Our new algorithm for dynamic clique maintenance is presented in Algorithm 4, and explained in the next example.

Example 2. Consider again Figure 2. We immediately update the graph $G = (V, E)$ with the set of new edges $\{\{3,4\},\{3,5\}\}$ (line 6). The set U becomes $\{3,4,5\}$ and $\text{fa}(U, G')$ actually equals V and so we will run Bron-Kerbosch on the whole graph (of course, for larger graphs this is rarely the case). Then we iterate through the existing cliques and remove those that intersect with U (line 9-13)—this step only leaves the clique $\{1,2\}$ in $\mathcal{C}$. Then we add all the cliques found in the subgraph $G'[\text{fa}(U, G')]$ (line 14-18) if and only if they intersect with U—in this case only $\{1,2\}$ is not added. We can observe that the clique $\{2,4,5\}$ is both removed and added again by the algorithm.

As the above example explains, our algorithm sometimes removes and adds the same clique. This is usually not a problem in practice, as comparison of cliques is much faster than calling `IsClique(·)`. The correctness of the algorithm follows from the results below.

Lemma 1. *Let* $G = (V, E)$ *be an undirected graph, and let* $G' = (V, E \cup F)$ *be the graph resulting from adding a set of new edges* F *to* G. *Let* $U = \mathcal{V}(F)$. *If* $C \in \mathcal{NC}(G, G')$, *then* $C \subseteq \text{fa}(U, G')$.

Proof. Since C is a new clique, it must contain at least two vertices from U. Since C is complete all vertices $v \in C \setminus U$ must be connected to some vertex in U. Hence v is a neighbour of U. □

Lemma 2. *Let* G *and* G' *be given as in Lemma 1. Then* $C \in \mathcal{RC}(G, G')$ *if and only if there exists* $K \in \mathcal{NC}(G, G')$ *such that* $C \subset K$.

Algorithm 4 Incremental clique maintenance by local search

```
1: function SetBasedUpdate(C, G, F)
2:     Input: A graph G = (V, E),
3:            the set of cliques C of G, and
4:            the set of new edges F.
5:     Let U = V(F)
6:     Let G' = (V, E ∪ F)
7:     Let C^new =
8:         BKWithPivot(G', ∅, fa(U, G'), ∅)
9:     for all C ∈ C do ▷ Remove old cliques
10:        if C ∩ U ≠ ∅ then
11:            Set C = C \ {C}
12:        end if
13:    end for
14:    for all C ∈ C^new do ▷ Add new cliques
15:        if C ∩ U ≠ ∅ then
16:            Set C = C ∪ {C}
17:        end if
18:    end for
19:    return C
20: end function
```

Proof. By Lemma 1, all new cliques $K \subseteq fa(U, G')$. Since a newly formed clique K is the only way to remove an existing clique C from $\mathcal{C}(G')$, the result follows. □

Theorem 2. *Let* G, G', F *and* U *be given as in Lemma 1. The cliques of* $\mathcal{C}(G')$ *can be found by removing the cliques from* $\mathcal{C}(G)$ *that intersect with* U *and adding cliques of* $G'[fa(U, G')]$ *that intersect with* U.

Proof. We first show we add all new cliques by considering just $G'[fa(U, G')]$. By Lemma 1, this subgraph contains all the new cliques. Furthermore, any new clique C must intersect with U; otherwise it could not contain a new edge. Therefore all the new cliques are added.

We remove all relevant cliques if $C \in \mathcal{RC}(G, G')$ implies $C \cap U \neq \emptyset$. So let $C \in \mathcal{RC}(G, G')$. Assume $C \cap U = \emptyset$; then for each $v \in nb(C, G) \cap U$, $C \not\subseteq nb(v, G)$ (otherwise C could not be a clique in G). But then no new clique can cover C which is a contradiction to Lemma 2. Hence $C \cap U \neq \emptyset$, and therefore we remove all relevant cliques.

Last we consider that we might also remove a clique $C \in \mathcal{C}(G) \cap \mathcal{C}(G')$. Since $C \cap U \neq \emptyset$, then $C \subseteq fa(U, G')$, and C will be added again when we add cliques from $G'[fa(U, G')]$ that intersect with U. □

Remark. Theorem 2 also implies that we can apply further pruning based on the set U in Algorithm 3. In particular, the for-loop in line 7 can be skipped if $(R \cup P) \cap U = \emptyset$. We have not implemented this pruning, however.

Remark. Stix derives a second approach to deal with the decremental problem. A nice property of our approach is that it works almost unaltered for this case (simply remove the set F of edges from the graph instead of adding them).

4 Triangulation by Clique Maintenance

An undirected graph is *triangulated* (or chordal) if every cycle of length greater than 3 has a chord. For example, the graphs in Figure 1 (left) and Figure 2 (right) are not triangulated, whereas the graph in Figure 1 (right) is triangulated. Triangulated graphs appears in remarkably diverse set of applications, ranging from efficient Gaussian elimination and compression in databases to compilation of Bayesian networks and decision graphs. It is the latter topic we have in mind in the following discussion. First we need some additional definitions.

The *elimination* of a vertex $v \in V$ of $G = (V, E)$ is the process of removing v from G and making $nb(v, G)$ a complete set. This process induces a new graph $H = (V \setminus \{v\}, E \cup F)$ where F is a set of *fill-in edges*. An *elimination order* of $G = (V, E)$ is a bijection $f : V \to \{1, 2, \ldots, |V|\}$ prescribing an order for eliminating all vertices of G. The *table size* of a clique C is given by $ts(C) = \prod_{v \in C} |sp(v)|$ where sp(v) denotes the state space of the variable corresponding to v in the Bayesian network. Finally, the *total table size* of a graph H is given by $tts(H) = \sum_{C \in \mathcal{C}(H)} ts(C)$.

Triangulation algorithms aim at minimizing different criteria. Two common criteria are the treewidth and the total table size criteria. The *treewidth* of a graph is the size of the largest

clique minus one, and the treewidth criterion requires the triangulated graph to have minimum treewidth. The total table size criteria requires the triangulated graph to have the minimum total table size. Of the two, the total table size criterion yields the most exact bound on the time and memory requirement of the subsequent inference in Bayesian networks (in particular when the domains of the variables have different size). In this section we shall therefore discuss how one may exploit knowledge of the cliques of a graph to perform a triangulation with minimum total table size. (This is an NP-hard problem (Wen, 1990), but for probabilistic inference we are in the fortunate situation that the task can often be performed off-line).

It is well-known that if all vertices are eliminated in G according to an elimination order f, the union of all the fill-in edges produced induces a triangulation of G. In this way each triangulation T of G corresponds to at least one elimination order, and we may explore the space of all possible triangulations $\mathcal{T}(\mathrm{G})$ by investigating all possible elimination orders. Even though the search space is of size $O(|\mathrm{V}|!)$, coalescing applies and reduces the search space to $O(2^{|\mathrm{V}|})$ size which turns out to be tractable for medium-sized models (say $|\mathrm{V}| \leq 64$) (Ottosen and Vomlel, 2010).

To perform this exploration, we may generate the search graph dynamically (on-demand) where each node corresponds to a subset of V being eliminated from G, and where each edge corresponds to a particular vertex being eliminated. (We use the term "node" exclusively for vertices in the search graph, and the term "vertices" exclusively for vertices in the graph being triangulated.) In the *start node* s no vertices have been eliminated, and in a *goal node* t all vertices have been eliminated and the graph G has been triangulated. To compute a cost for each path in the search graph, we associate the following with each node n:

1. $\mathrm{H} = (\mathrm{V}, \mathrm{E} \cup \mathrm{F})$: the original graph with all fill-in edges F accumulated along the path to n from the start node s.
2. The set of cliques for H, $\mathcal{C}(\mathrm{H})$.
3. The total table size for the graph H.

To maintain tts(H) efficiently we need $\mathcal{C}(\mathrm{H})$ which in turn requires H (as described in Section 3). The following example shows how one particular path in the search space is generated.

Example 3. Consider the graphs in Figure 3 where fill-in edges are indicated with dotted lines. We follow an elimination order starting with $\langle 6, 4 \rangle$. The cliques of the initial graph may be computed using the Bron-Kerbosch algorithm (Algorithm 3), and the total table size is (assuming binary variables) $3 \cdot 2^2 + 2 \cdot 2^3 = 28$ which is an lower estimate of the total table size of the triangulated graph. This information is then associated with the start node s.

Then we can make vertex 6 simplicial and run the clique update algorithm such the set of cliques is up-to-date. Again we can recompute the total table size. This information is then associated with a successor node n of s and the edge between them is labelled with vertex 6.

When we generate a successor node m of n (corresponding to the elimination of vertex 4), we must not add fill-ins to already eliminated vertices. Therefore the relevant graph corresponds to Figure 3 (right) including the fill-in edge (in particular, $\{2, 6\}$ should not be a fill-in edge). In this manner one may continue until the graph is triangulated. In this case, this happens after we eliminate vertex 4. Finally, we see that the cliques of the triangulated graph are $\{3, 4, 5, 6\}$, $\{2, 3, 4, 5\}$, and $\{1, 2, 3\}$, thus its true total table size equals $2 \cdot 2^4 + 2^3 = 40$.

5 Results

In this section we shall compare the two approaches for dynamic clique maintenance. For that purpose we have used a set of public Bayesian networks[1]. This gave us 7 real-world undirected graphs, and for each graph we generated 10 more by successively adding 5% of the missing (undirected) edges at random (in total 77 graphs). We have then performed two tests on this dataset:

[1] `http://compbio.cs.huji.ac.il/Repository/networks.html`

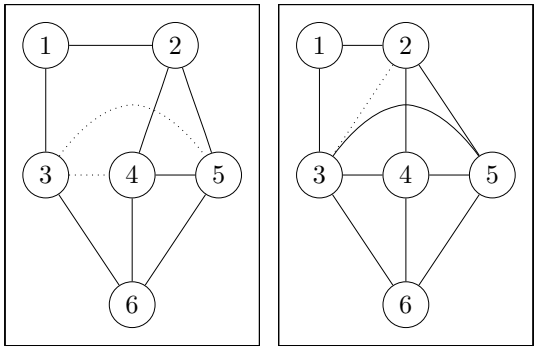

Figure 3: Left: The initial graph $G = (V, E)$: when we eliminate vertex 6, we need to add the fill-ins $\{3, 4\}$ and $\{3, 5\}$. Right: The graph after adding fill-in edges induced by eliminating vertex 6: when we eliminate vertex 4, the fill-in $\{2, 3\}$ is added and the graph is now triangulated.

1. For each graph in the dataset, add all the missing edges in isolation. The set of cliques is updated after an edge is added. Then the edge is removed, and the next edge is added etc.
2. For each graph in the dataset, triangulate the graph by making all vertices simplicial in some random order. The set of cliques is updated after each vertex is made simplicial. Already simplicial vertices are removed before an update. (Note that we did not moralize the initial directed network even though this might lead to a slightly more accurate test.)

These two test scenarios were chosen because we believe that they show the worst-case performance (scenario 1), and the expected speedup for our triangulation problem (scenario 2). For each graph we then ran the tests 1000 times (with different random order each time for Test 2) and saved the mean time. We have then plotted the mean time of Stix' approach divided by the mean time of the new approach (we call this the "saving ratio").

In Figure 4 and 5 we have collected the results of the two tests. The total time saving ratio for various graph densities are summarized in Table 1. We can see that even for Test 1, the new

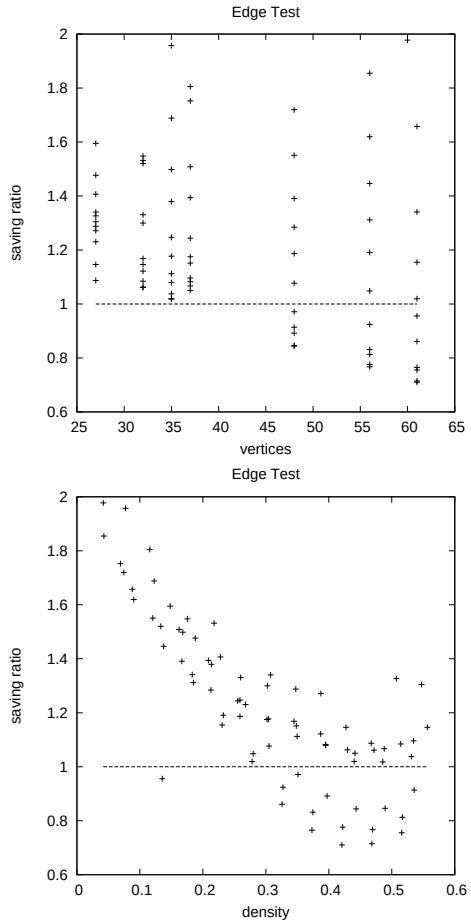

Figure 4: Results for Test 1: updating the set of cliques after adding a single edge. Points above $y = 1$ indicate that the new approach is faster.

method often performs better. However, overall Stix' method works better for more dense graphs. There seems to be no clear connection between the size of the graph and saving ratio.

For Test 2 the new method is significantly better, especially for more dense graphs. There also seems to be a connection between the size of the graph and performance, with the saving ratio increasing as the size increases. This might be because larger graphs allow for more cliques and larger neighbour sets.

6 Conclusion

We have described a new method for maintaining the cliques of a dynamic graph. The new method works by employing a local search for cliques—the local search can in principle

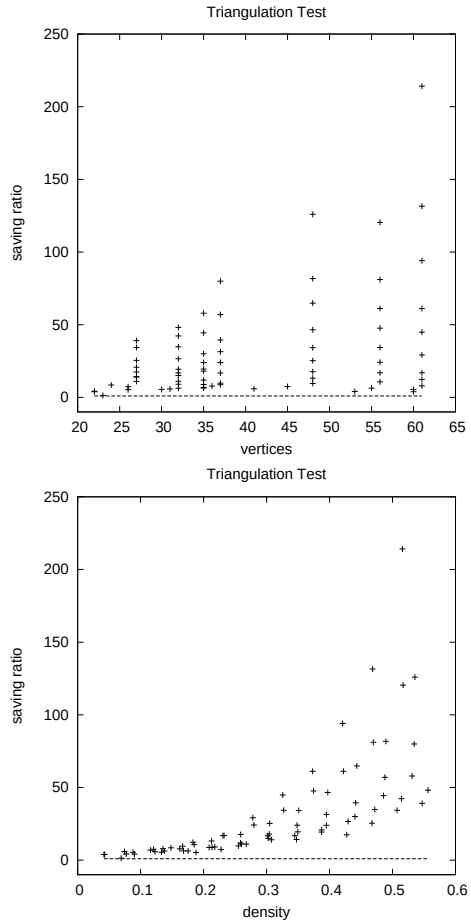

Figure 5: Results for Test 2: updating the set of cliques after adding fill-in edges. Points above $y = 1$ indicate that the new approach is faster.

be done by any existing clique search algorithm. The new method is both simpler and more generic than previous methods, and experiments show that the new method performs significantly faster when adding a set of fill-in edges.

We also described how dynamic clique maintenance algorithms may found the basis for new total table size triangulations methods. These methods may be optimal off-line triangulations, or they may be any-time triangulation heuristics. Since the off-line triangulations methods can explore the whole space of possible triangulations, they can also be used to give a precise evaluation of the quality of the triangulations returned by heuristics.

Table 1: Total time saving ratio for different graph densities. A value above 1 indicates that the new method was faster in terms of total running time for all graphs of the specified density.

Density δ	Test 1	Test 2
$\delta \in [0, 0.1)$	1.74	4.77
$\delta \in [0.1, 0.2)$	1.32	9.09
$\delta \in [0.2, 0.3)$	1.12	19.35
$\delta \in [0.3, 0.4)$	0.88	40.95
$\delta \in [0.4, 0.5)$	0.76	86.68
$\delta \in [0.5, 0.6)$	0.80	153.04

Acknowledgements

We would like to thank the three anonymous reviewers for their constructive comments.

J. Vomlel was supported by the Ministry of Education of the Czech Republic through grants 1M0572 and 2C06019 and by the Czech Science Foundation through grants ICC/08/E010 and 201/09/1891.

References

F. Cazals and C. Karande. 2008. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3):564–568, November.

R. M. Karp. 1972. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

Ina Koch. 2001. Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci.*, 250(1-2):1–30.

J. W. Moon and L. Moser. 1965. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28.

Thorsten J. Ottosen and Jiří Vomlel. 2010. All roads lead to Rome—new search methods for optimal triangulations. In *Proceedings of the Fifth European Workshop on Probabilistic Graphical Models*.

Volker Stix. 2004. Finding all maximal cliques in dynamic graphs. *Comput. Optim. Appl.*, 27(2):173–186.

Wilson Wen. 1990. Optimal decomposition of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pages 209–224, New York, NY. Elsevier Science.