Maximal Matching and Path Matching Counting in Polynomial Time for Graphs of Bounded Clique Width

Benjamin Hellouin de Menibus[1] and Takeaki Uno[2]

[1] ENS Lyon, France, benjamin.hellouin_de_menibus@ens-lyon.fr
[2] National Institute of Informatics, Tokyo 101–8430, Japan, uno@nii.jp

Abstract. In this paper, we provide polynomial-time algorithms for different extensions of the matching counting problem, namely maximal matchings, path matchings (linear forest) and paths, on graph classes of bounded clique-width. For maximal matchings, we introduce matching-cover pairs to efficiently handle maximality in the local structure, and develop a polynomial time algorithm. For path matchings, we develop a way to classify the path matchings in a polynomial number of equivalent classes. Using these, we develop dynamic programing algorithms that run in polynomial time of the graph size, but in exponential time of the clique-width. In particular, we show that for a graph G of n vertices and clique-width k, these problems can be solved in $O(n^{f(k)})$ time where f is exponential in k or in $O(n^{g(l)})$ time where g is linear or quadratic in l if an l-expression for G is given as input.

Keywords: maximal matching, path matching, counting, clique-width.

1 Introduction

Counting problems in graphs can be very difficult, i.e. $\#P$-hard in the general case, even for simple objects such as trees and independent sets. Research on graph classes has been motivated by such "hard" decision or optimization problems, and restricting the input to given graph classes has led to numerous polynomial-time algorithms. Despite this, only a few useful algorithms for counting problems exist, and these are relatively recent.

In this paper, we focus on counting maximal matching and path matching (linear forest). Matching counting and all extensions considered in this paper have been proved $\#P$-complete in the general case. Some sparse graph classes such as planar graphs or graphs of bounded tree-width allow polynomial-time algorithms for perfect matching counting (see [13] and [1]); on the negative side, Valiant, when introducing the class $\#P$, proved that counting perfect matchings as well as general matchings in bipartite graphs is $\#P$-complete [19, 20]. Valiant's proof concerning matchings has since been extended to 3-regular bipartite graphs [8], bipartite graphs of maximum degree 4 and bipartite planar graphs of maximum degree 6 [18].

The problem of counting perfect matchings in chordal and chordal bipartite graphs is also $\#P$-complete [16], but good results on independent sets [15] give the impression that the chordal structure could nevertheless be interesting regarding matching counting. This led us to focus on a related graph class, the $(5, 2)$-crossing-chordal graphs. We especially make use of the bounded clique-width of this graph class.

Courcelle et al. introduced clique-width in [5] as a generalization of tree-width, and it attracted attention mainly for two reasons. On the one hand, in a similar fashion as the tree width, putting a bound on the clique-width makes many difficult problems decidable in polynomial time (see for example [6]). On the other hand, this class contains dense graphs as well as sparse graphs, which leads to more general results.

Makowsky et al. already proved as a consequence of a result in [14] that matching counting on graphs of bounded clique-width is polynomial in the size of the input graph. In this paper, we will extend this result by adapting their method to maximal matchings and path matchings. Our algorithms are polynomial of the graph size, but exponential of the clique-width k, i.e., $O(n^{poly(k)})$ time. It might be hard to develop a fixed parameter tractable algorithm such as an $O(c^{poly(k)}poly(n))$ time algorithm, since many graph algorithms, e.g. vertex coloring, have to spend $O(n^{poly(k)})$ time unless FPT $\neq W[1]$ [10].

The existing matching counting algorithms can not be used to count maximal matchings directly. The algorithms in [14] classify matchings of local graphs according to their sizes and the colors of the endpoints, and then get information about larger graphs by merging the matchings. However, in this way, each classified group may contain both matchings included in maximal matchings and those not included in any maximal matching. Actually, it seems to be difficult to characterize the number of matchings included in some maximal matching, by using only their sizes and their endpoints. In this paper, we introduce matching-cover pairs for this task. When we restrict a maximal matching to a subgraph, it can be decomposed into the matching edges belonging to the subgraph and end vertices of matching edges not included in the subgraph. From the maximality, the end vertices form a vertex cover of the edges of the subgraph. Thus, we count such pairs of matching and vertex cover according to their sizes and colors, and obtain a polynomial time algorithm for the problem.

For the problem of counting paths and path matchings, we must have some way to handle the connectivity of edge sets. Actually, connectivity is not easy to handle; for example, checking for the existence of Hamiltonian path is equivalent to checking whether the number of paths of length $n-1$ is larger than zero or not. Gimenez et al. devised an algorithm based on Tutte polynomial computation to count the number of forests in bounded-clique-width graphs in sub-exponential time, running in $2^{O(n^c)}$ time for constant $c < 1$ [11]. We use the properties of bounded-clique-width graphs so that we can classify the path matchings in a polynomial number of groups of equivalent path matchings, and thereby compute the number of paths and path matchings in polynomial time.

2 Clique Width

We shall introduce clique-width on undirected, non-empty labeled graphs by a construction method. Let G_i be the subgraph of vertices labeled i in a graph G. We define the singleton S_i as the labeled graph with one vertex of label i and no edge, and the following construction operations:

- Renaming : $\rho_{i\to j}(G)$ is G where all labels i are replaced by labels j;
- Disjoint union : $(V_1, E_1) \oplus (V_2, E_2) = (V_1 \cup V_2, E_1 \cup E_2)$;
- Edge creation : $\eta_{i,j}((V, E)) = (V, E \cup \{(v_1, v_2) \mid v_1 \in G_i, v_2 \in G_j\})$.

The class of graphs with clique-width $\leq k$ is the smallest class containing the singletons S_i, closed under $\rho_{i\to j}, \oplus$ and $\eta_{i,j}$ $(1 \leq i, j \leq k)$. In other words, the *clique-width* of a graph G, denoted as $cwd(G)$, is the minimal number of labels necessary to construct G by using singletons and renaming, disjoint union and edge creation operations.

For an unlabeled graph G, we define its clique-width by labeling all vertices with label 1. This is necessarily the best labeling, since any labeling can be renamed to a monochromatic labeling. Note that the clique-width of a graph of order n is at most n.

$(5, 2)$-crossing-chordal graphs are known to have clique-width ≤ 3 [3] (we recall that a $(5, 2)$-crossing-chordal graph is a graph where any cycle of length ≥ 5 has a pair of crossing diagonals). Other interesting results include: cographs are exactly the graphs with $cwd(G) \leq 2$, planar graphs of bounded diameter have bounded clique-widths, and any graph class of treewidth $\leq k$ also has a bounded clique-width of $\leq 3.2^{k-1}$ [4]. A complete review can be found in [12].

An *l-expression* is a term using $S_i, \rho_{i\to j}, \eta_{i,j}$ and $\oplus$ (with $i, j \leq l$) that respects the arity of each operation. It can be represented more conveniently in a tree structure, and we can inductively associate the current state of the construction with each node. If G is the graph associated with the root, we say that this term is an l-expression for G, and it is a certificate that G is of clique-width $\leq l$. An example is given in Fig.1.

Fellows et al. proved the NP-hardness of computing the minimum clique-width for general graphs [9]. The current best approximation is due to Oum and Seymour [17], who provided a linear time algorithm that, given a graph G and an integer c as input, returns an 2^{3c+2}-expression for G or certifies that the graph has a clique-width larger than c.

This implies that we can compute in quadratic time a 2^{3k+2}-expression for a graph of clique-width k by applying this algorithm for $c = 1, 2 \ldots$. As the bound is independent of n, algorithms requiring expressions as input will still work in polynomial time, although the time complexity will usually be extremely poor. For $(5, 2)$-crossing-chordal graphs, though, this is not a concern since it is possible to compute a 3-expression in linear time [3].

An l-expression is called *irredundant* if every edge-creation operation $\eta_{i,j}$ is applied to a graph where no two vertices in G_i and G_j are adjacent. Any

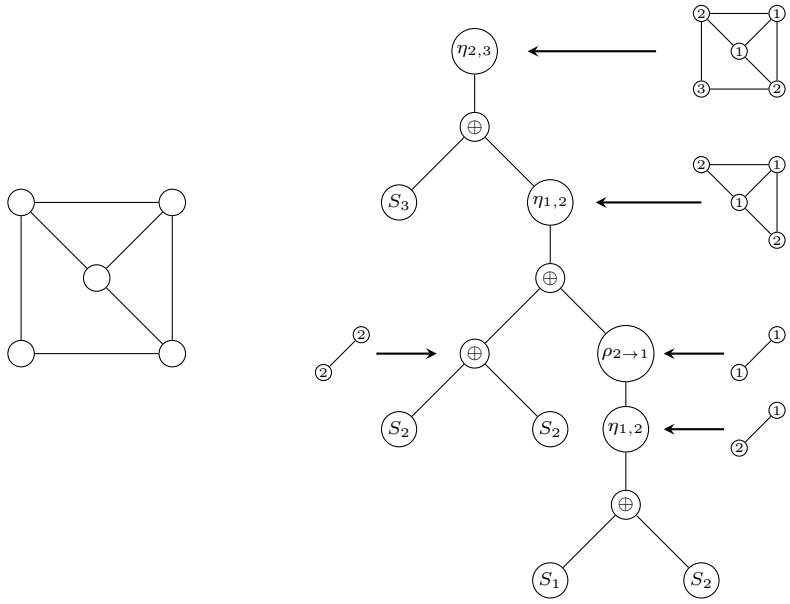

Fig. 1. Graph of clique-width 3, and a possible 3-expression tree (the last renaming operations are omitted).

l-expression can be turned into an l-irredundant expression in linear time [7]. Therefore, we assume w.l.o.g. that the input expression is irredundant.

3 Framework of Our Algorithms

The input of our algorithms is a graph G on n vertices and an l-expression for G, and the output is the number of objects (ex. matchings, paths) in G. The procedure works by counting these objects at each step of the construction, by using the expression tree : we start from the leaves and process a node once all its children have been processed. Finally, the value at the root of the tree is the output of the algorithm. Instead of doing it directly with the considered object, we introduce appropriate intermediate objects, and we compute tables of values at each step.

To avoid tedious case studies, we shall assume that requesting the value of any vector outside of the range $\{0 \dots n\}$ returns the value 0. Also, $\Delta_r(l)$ is the vector $(\delta_{i,r})_{1\leq i\leq l}$, and $\Delta_{r,s}(l)$ is the vector $(\delta_{i,r}\cdot\delta_{j,s})_{\substack{0\leq i\leq j\leq l\\(i,j)\neq(0,0)}}$, where $\delta_{i,j}$ is the *Kronecker delta*:

$$\delta_{i,j} = \begin{cases} 1 \text{ if } i = j; \\ 0 \text{ otherwise.} \end{cases} \tag{1}$$

We will omit the l when it is obvious from the context.

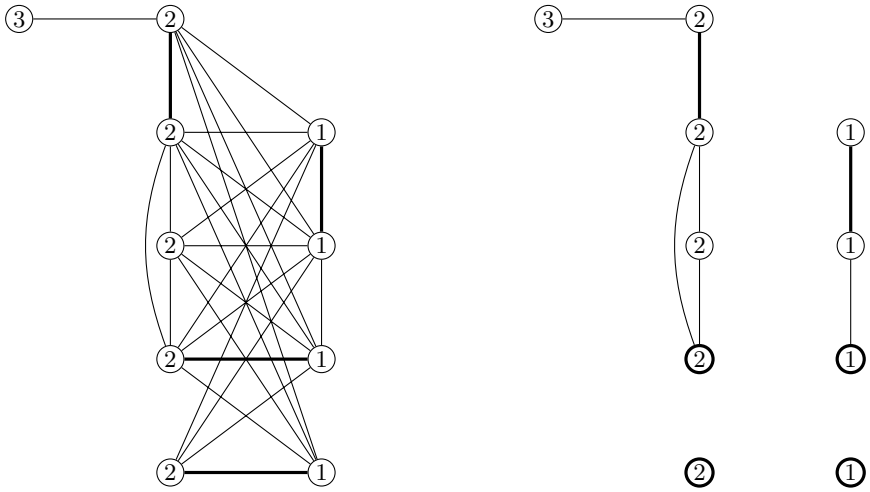

Fig. 2. Maximal matching of $\eta_{1,2}(G')$, and the corresponding matching-cover pair of G'.

4 Counting Maximal Matchings

Theorem 1. *Computing the number of maximal matchings of a graph with n vertices with a corresponding l-expression can be done in polynomial time in n (but exponential w.r.t l).*

We cannot directly use the previous framework on maximal matchings. Indeed, consider M a maximal matching of $G = \eta_{i,j}(G')$ and M' the induced matching in G': M' is not necessarily maximal. However, we can keep track of the vertices of G' that are covered in M, and those vertices must form a vertex cover of the subgraph left uncovered by M'. See Fig.2 for an example.

A *matching-cover* pair of a graph $G = (V, E)$ is a pair (m, c) such that:

- $m \subseteq E$ is a matching of G (i.e. no vertex is covered more than once);
- $c \subseteq V$ is a vertex cover of the subgraph left uncovered by m (i.e. every *edge* is covered at least once).

We show that computing the number of matching-cover pairs of a graph with n vertices with a corresponding l-expression can be done in polynomial time in n.

Let $M = (m_i)_{1 \leq i \leq l}$ and $C = (c_i)_{1 \leq i \leq l}$ be two vectors of non-negative integers. For a graph G, we say that a pair (m, c) satisfies the condition $\varphi_{M,C}(G)$ if m covers m_i vertices in G_i and c uses c_i vertices in G_i for all i, and we denote by $mc_{M,C}(G)$ the number of pairs that satisfy $\varphi_{M,C}(G)$. Note that maximal matchings are exactly pairs with an empty cover; therefore, the number of maximal matchings of G is $\sum_{k \leq n} mc_{k \cdot \Delta_1, 0}(G)$.

Now we will follow the framework described above and compute $mc_{M,C}$ for all possible M and C, at each step of the construction. We associate to each node of the tree a table of size n^{2l} corresponding to the values of $mc_{M,C}$ on this graph for M and C ranging from $(0,..,0)$ to $(n,..,n)$. For a singleton S_i, we can easily see that:

$$mc_{M,C}(S_i) = \begin{cases} 1 \text{ if } M = 0 \text{ and } C = 0 \text{ or } \Delta_i; \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

For the renaming operation $G = \rho_{i \to j}(G')$, the graph is not modified, but all vertices of label i are set to label j. Hence, we modify the entries i and j accordingly.

$$mc_{M,C}(G) = \sum_{\substack{M':(M,M') \vdash \phi_{i,j} \\ C':(C,C') \vdash \phi_{i,j}}} mc_{M',C'}(G') \tag{3}$$

$$\text{where } (X, X') \vdash \phi_{i,j} \Leftrightarrow \begin{pmatrix} x_j = x'_i + x'_j \\ x_i = 0 \\ \forall k \notin \{i,j\}, x_k = x'_k \end{pmatrix} \tag{4}$$

For the disjoint union of two graphs $G = G_1 \oplus G_2$, we have a bijection between matching-cover pairs (m, c) in G and pairs $(m_1, c_1), (m_2, c_2)$ of matching-cover pairs in G_1 and G_2, respectively. Moreover, if (m, c) satisfies $\varphi_{M,C}$, (m_1, c_1) satisfies φ_{M_1,C_1} and (m_2, c_2) satisfies φ_{M_2,C_2}, we have $M = M_1 + M_2$ and $C = C_1 + C_2$.

$$mc_{M,C}(G) = \sum_{\substack{M_1+M_2=M \\ C_1+C_2=C}} mc_{M_1,C_1}(G_1) \cdot mc_{M_2,C_2}(G_2) \tag{5}$$

For the edge creation operation $G = \eta_{i,j}(G')$, we have to choose the extremities of the edges added to the matching among the vertices in the vertex cover. If q is the number of new edges, we have:

$$mc_{M,C}(G) = \sum_{q=0}^{n} mc_{M',C'}(G') \cdot \binom{c'_i}{q} \cdot \binom{c'_j}{q} \cdot q! \tag{6}$$

$$\text{where } M' = M - q\Delta_i - q\Delta_j,\ C' = C + q\Delta_i + q\Delta_j \tag{7}$$

Once the maximal matchings of all sizes are computed, it is straightforward to count the number of perfect matchings and the number of minimum maximal matchings in polynomial time. Note that counting perfect matchings can be achieved in $O(n^{2l+1})$ time by adapting the matching counting algorithm presented in [14] in a similar fashion.

Complexity study: Obviously, there are exactly n singleton operations, and each operation requires a constant amount of time. Every other operation requires one to compute n^{2l} values. As the expression is irredundant, every edge

creation operation adds at least one edge, so there are at most n^2 edge creation operations, processed in linear time. As a disjoint union operation has two children in the tree, and there are n leaves, there are $n-1$ disjoint union operations, and they require $O(n^{2l})$ time.

For the renaming operation, consider the number of different labels at each step of the construction. This number is one for a singleton, the edge creation operation has no effect, the disjoint union is an addition in the worst case (no shared label) and the renaming operation diminishes this number by one. Therefore, there are at most n renaming operations, and they are done in $O(n^4)$ time. The final sum requires $O(n^l)$ operations.

Therefore, the overall complexity of the algorithm is

$$O(n) + O(n^{2l}) \cdot \big(O(n^5) + O(n^{2l+1}) + O(n^3)\big) + O(n^l) = O(n^{4l+1}) \ (l \geq 2). \quad (8)$$

For $(5,2)$-crossing-chordal graphs, we can compute an expression of width $l = 3$ in linear time and the algorithm runs in time $O(n^{13})$.

5 Counting paths and path matchings

A *path matching* (or *linear forest*) is a disjoint union of paths, in other words, a cycle-free set of edges such that no vertex is covered more than twice.

Theorem 2. *Computing the number of paths $pth(G)$ and the number of path matchings $pm(G)$ of a graph of clique-width $\leq k$ can be done in polynomial time (but exponential w.r.t. k).*

Proof. Let $K = (k_{i,j})_{\substack{0 \leq i \leq j \leq l \\ (i,j) \neq (0,0)}}$ be a vector of non-negative integers. We say that a path matching P of G satisfies the condition ψ_K if:

- $\forall i > 0, k_{0,i}$ vertices in G_i are left uncovered by P;
- $\forall (i,j), i \leq j$, $k_{i,j}$ paths in P have extremities in G_i and G_j.

We denote the number of path matchings in G satisfying ψ_K by $pm_K(G)$. If $i > j$, we denote $k_{i,j} = k_{j,i}$. As K is of size $\frac{l(l+3)}{2}$, we compute tables of size $n^{\frac{l(l+3)}{2}}$ at each step.

For a singleton S_i, the only possible path matching is empty and leave the vertex uncovered.

$$\forall K, pm_K(S_i) = \begin{cases} 1 \text{ if } K = \Delta_{0,i}; \\ 0 \text{ otherwise.} \end{cases} \quad (9)$$

For the renaming operation $G = \rho_{i \to j}(G')$, the method is the same as for maximal matchings.

$$pm_K(G) = \sum_{K' : (K,K') \vdash \phi} pm_{K'}(G') \quad (10)$$

$$\text{where } (K,K') \vdash \phi \Leftrightarrow \begin{pmatrix} k_{j,j} = k'_{j,j} + k'_{i,j} + k'_{i,i} \\ \forall a \notin \{i,j\}), k_{a,j} = k'_{a,i} + k'_{a,j} \\ \forall a, k_{a,i} = 0 \\ \forall a \notin \{i,j\}, b \notin \{0,i,j\}, k_{a,b} = k'_{a,b} \end{pmatrix} \tag{11}$$

For the disjoint union operation $G = G_1 \oplus G_2$, we have a bijection between path matchings p in G and pairs (p_1, p_2) of path matchings in G_1 and G_2, respectively. Plus, if p_1 satisfies ψ_{K_1}, p_2 satisfies ψ_{K_2} and p satisfies ψ_K, we have $K = K_1 + K_2$.

$$pm_K(G) = \sum_{K_1+K_2=K} pm_{K_1}(G_1) \cdot pm_{K_2}(G_2) \tag{12}$$

Consider now the edge creation operation $G = \eta_{i,j}(G')$. We say a path matching P in G is an *extension* of a path matching P' in G' if $P \cap G' = P'$, so that $P = P' \cup E_{i,j}$ where $E_{i,j}$ is a subset of the edges added by the operation. Now, if we consider a path matching P' in G' that satisfies $\psi_{K'}$, we claim that the number of extensions of P' in G that satisfy ψ_K depends only on i, j, K' and K (and not on P' or G'), and we represent it as $N_{i,j}(K, K')$. Since every path matching of G is an extension of an unique path matching of G', we have:

$$pm_K(G) = \sum_{K'} pm_{K'}(G') \cdot N_{i,j}(K', K) \tag{13}$$

Moreover, we can compute all the $N_{i,j}(K', K)$ beforehand in $O(n^{l(l+4)})$ time. The proof of these claims is given in the appendix.

We can then compute the number of paths $pth(G)$ and the number of path matchings $pm(G)$ with the formulas:

$$\begin{aligned} pth(G) &= \sum_{0 \le a \le n} pm_{K(a)}(G) \quad \text{where } K(a) = a \cdot \Delta_{0,1} + \Delta_{1,1} \\ pm(G) &= \sum_{1 \le a+2b \le n} pm_{K(a,b)}(G) \text{ where } K(a,b) = a \cdot \Delta_{0,1} + b \cdot \Delta_{1,1} \end{aligned} \tag{14}$$

Complexity study: A singleton operation requires constant time. Every other operation requires us to compute $n^{\frac{l(l+3)}{2}}$ values. For each value, the renaming operation in processed in linear time, the disjoint union operation in $O(n^{l^2})$ time and the edge creation operation in $O(n^{\frac{l(l+3)}{2}})$ time.

The overall complexity of the algorithm is:

$$\begin{cases} O(n^{l^2+4l}) \text{ for } l \le 5; \\ O(n^{\frac{3}{2}(l^2+l)+1}) \text{ for } l > 5. \end{cases} \tag{15}$$

For $(5,2)$-crossing-chordal graphs, we can compute in linear time an expression of width $l = 3$ and we have an algorithm running in $O(n^{21})$ time.

6 Conclusion

These results seem to confirm the intuition that bounding clique-width is an efficient restriction on the input of $\#P$-hard problems in order to allow the use of polynomial algorithms. Notably, being able to count paths and path matchings in polynomial time is interesting because connected structures are usually very difficult to count. In that sense, the next logical step was to study the tree (or, equivalently, forest) counting problem. However, our attempts to do so by using a method similar to the one we used in the paper, only produced algorithms running in exponential time. Our feeling is that the tree counting problem remains $\#P$-complete for graphs of bounded clique-width, as this intuitive method keeps giving bad results. It remains an open problem for now.

References

1. S. Arnborg, J. Lagergren and D. Seese: Easy Problems for Tree-Decomposable Graphs, Journal of Algorithms, **12**, pp. 308–340, 1991.
2. H. J. Bandelt and H. M. Mulder: Distance-hereditary Graphs, Journal of Combinatorial Theory, Series B, **41**(2), pp. 182–208, 1986.
3. D. G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed and U. Rotics: Polynomial Time Recognition of Clique-Width ≤ 3 Graphs, Proceedings of the 4th Latin American Symposium on Theoretical Informatics, Lecture Notes in Computer Science, **1776**, pp. 126–134, 2000.
4. D. G. Corneil and U. Rotics: On the Relationship Between Clique-Width and Treewidth, SIAM Journal of Computing, **34**(4), pp. 825–847, 2005.
5. B. Courcelle, J. Engelfriet and G. Rozenberg: Handle-Rewriting Hypergraph Grammars, J. Comput. Syst. Sciences, **46**, pp. 218-270, 1993.
6. B. Courcelle, J. A. Makowsky and U. Rotics: Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width, Theory of Computing Systems, **33**, pp. 125–150, 2000.
7. B. Courcelle and S. Olariu: Upper Bounds to the Clique-Width of Graphs, Discrete Applied Mathematics, **101**, pp. 77–114, 2000.
8. P. Dagum and M. Luby: Approximating the Permanent of Graphs with Large Factors, Theoretical Computer Science, **102**, pp. 283–305, 1992.
9. M. R. Fellows, F. A. Rosamond, U. Rotics and S. Szeider: Clique width Minimization is NP-hard, Annual ACM Symposium on Theory of Computing, Proceedings of the 38th annual ACM symposium on Theory of computing, session 8B, pp. 354–362, 2006.
10. F. V. Fomin, P. A. Golovach, D. Lokshtanov and S. Saurabh: Algorithmic Lower Bounds for Problems Parameterized by Clique-Width, Proceedings of the 21th ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), ACM and SIAM, pp. 493–502, 2010.
11. O. Gimenez, P. Hlineny and M. Noy, Computing the Tutte Polynomial on Graphs of Bounded Clique-Width, *LNCS* **3787**, pp.59-68, 2005.
12. M. C. Golumbic and U. Rotics: On the Clique-Width of Some Perfect Graph Classes, International Journal of Foundations of Computer Science, **11**(3), pp. 423–443, 2000.
13. P. W. Kasteleyn: Dimer Statistics and Phase Transitions, Journal of Mathematical Physics, **4**, pp. 287–293, 1963.

14. J. A. Makowsky, U. Rotics, I. Averbouch and B. Godlin: Computing Graph Polynomials on Graphs of Bounded Clique-Width, Lecture notes in Computer Science, **4271**, pp. 191–204, 2006.
15. Y. Okamoto, T. Uno and R. Uehara: Counting the Independent Sets in a Chordal Graph, Journal of Discrete Algorithms, **6**(2), pp. 229-242, 2008.
16. Y. Okamoto, R. Uehara and T. Uno: Counting the Number of Matchings in Chordal and Chordal Bipartite Graph Classes, Proceedings of 35th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS, **5911**, pp. 296–307, 2009
17. S. Oum and P. Seymour: Approximating Clique-Width and Branch-Width, Journal of Combinatorial Theory, Series B, **96**(4), pp. 514–528, 2006.
18. S. P. Vadhan: The Complexity of Counting in Sparse, Regular, and Planar Graphs, SIAM Journal on Computing, **31**, pp. 398–427, 2001.
19. L. G. Valiant: The Complexity of Computing the Permanent, Theoretical Computer Science, **8**, pp. 189–201, 1979.
20. L. G. Valiant: The Complexity of Enumeration and Reliability Problems, SIAM Journal on Computing, **8**, pp. 410–421, 1979.

Appendix

We now prove the case of Thm. 2 we have omitted. Let G and G' be two labeled graphs such that $G = \eta_{i,j}(G')$ (for some $i < j$) and P' a path matching of G' satisfying $\psi_{K'}$ for some K'. For any K, we want to compute the number of extensions of P' in G satisfying ψ_K.

Definitions. For any path matching satisfying ψ_K, a path with two extremities $x \in G_i$ and $y \in G_j$ is called an (i,j)-*path*, and x and y are called *partners*. We denote by $V_a(b)$ the vertices of G_a whose partner is in G_b, and by $V_a(0)$ the uncovered vertices in G_a. We also note $v_{a,b} = \#V_a(b)$, which means that $v_{a,b} = k_{a,b}$, except for $v_{a,a} = 2k_{a,a}$ (note that $v_{a,b}$ depend only on K). An edge which extremities are in $V_i(a)$ and in $V_j(b)$, respectively, is called an (a,b)-*edge*.

We use a dynamic programming technique to build all possible extensions of P' by considering each vertex of G'_i one by one in the order $V_i(j), V_i(0), .., V_i(l)$ (the reason for this order will be explained later). If $X = (x_0, .., x_l)$ is a vector of non-negative integers and P_1 a path matching in G_1 that satisfies ψ_{K_1}, $T_{i,j}(G_1, P_1, K_2, X)$ stands for the number of extensions of P_1 in $\eta_{i,j}(G_1)$ that satisfy ψ_{K_2} and that uses only the x_k last vertices of $V_i(k)$ for every k.

At each step of the computation, the equations show that knowing G_1 and P_1 is not necessary as long as ψ_{K_1} is satisfied: this proves our first claim, and we write $T_{i,j}(K_1, K_2, X)$ for $T_{i,j}(G_1, P_1, K_2, X)$. Also, since i, j and K_2 are not modified during the computation, we write $T(K_1, X)$ for $T_{i,j}(K_1, K_2, X)$.

We now detail the different steps by increasing difficulty (instead of the actual order of the algorithm). First, assume that $x_j = x_0 = .. = x_{k-1} = 0$ and

$x_k \neq 0$ (for some $k \neq i$). We consider the first vertex in $V_i(k)$ that has not been considered yet in the computation. We have only two possibilities:

- No new edge adjacent to this vertex is added to the path matching.
- One new (k, a)-edge (possibly $a = 0$) is added to the path matching: we have $v_{j,a}$ choices for the edge. A (i, k)-path and a (j, a)-path are transformed into a (k, a)-path.

In each case, the value of the current K is updated accordingly and the vertex is deleted from X. Next to each term is the set that contains the other extremity of the edge being considered.

$$\begin{aligned} T(K", X) = T(K", X - \Delta_k) & \qquad \emptyset \\ + \sum_{1 \leq a \leq l} v_{j,a}.T(K" - \Delta_{i,k} - \Delta_{j,a} + \Delta_{k,a}, X - \Delta_k) & \qquad V_j(a) \\ + v_{0,j}.T(K" - \Delta_{i,k} - \Delta_{0,j} + \Delta_{j,k}, X - \Delta_k) & \qquad V_j(0) \end{aligned} \tag{16}$$

Note that if a (k, i)-edge is added, the partner of another vertex of $V_i(j)$ is also modified: this is why $V_i(j)$ is considered first in the computation, so that it does not appear in X anymore at this step. This remark holds for all the other cases except for $k = j$.

Now, we consider the first step ($k = j$). The situation is similar, but the vertex cannot be linked to its own partner when $k' = i$. Note that adding a (j, i)-edge changes the partner of another vertex of $V_i(j)$, but the new partner is still in G_j, so doing this brings no modification to X.

$$\begin{aligned} T(K", X) = T(K", X - \Delta_j) & \qquad \emptyset \\ + \sum_{\substack{1 \leq a \leq l \\ a \neq i}} v_{j,a} T(K" - \Delta_{i,j}, X - \Delta_j) & \qquad V_j(a) \\ + (v_{i,j} - 1) T(K" - \Delta_{i,j}, X - \Delta_j) & \qquad V_j(i) \\ + v_{0,j}.T(K" - \Delta_{i,j} - \Delta_{0,j} + \Delta_{j,j}, X - \Delta_j) & \qquad V_j(0) \end{aligned} \tag{17}$$

For the uncovered vertices ($k = 0$), up to two edges can be added to the path matching. The possibilities are:

- No new edge adjacent to this vertex is added to the matching.
- One new (k, k')-edge is added to the matching: we have $v_{j,k'}$ choices for the edge. An uncovered vertex and a (j, k')-path are transformed into a (x, k')-path.
- Two new (k, k') and $(k, k")$-edges are added to the matching: we have $v_{j,k'}.v_{j,k"}$ choices for the two edges (only half of those when $k' = k"$). An uncovered vertex, a (j, k')-path and a $(j, k")$-path are transformed into a $(k', k")$-path.

$$
\begin{aligned}
T(K", X) = T(K", X - \Delta_0) & \qquad \emptyset \\
+ \sum_{1 \le a \le l} v_{j,a}.T(K" - \Delta_{0,i} - \Delta_{j,a} + \Delta_{i,a}, X - \Delta_0) & \qquad V_j(a) \\
+ v_{0,j}.T(K" - \Delta_{0,i} - \Delta_{0,j} + \Delta_{i,j}, X - \Delta_0) & \qquad V_j(0) \\
+ \sum_{1 \le a < b \le l} v_{j,a}.v_{j,b}.T(K" - \Delta_{0,i} - \Delta_{j,a} - \Delta_{j,b} + \Delta_{a,b}, X - \Delta_0) & \qquad V_j(a) \mid V_j(b) \\
+ \sum_{1 \le a \le l} v_{j,a}.v_{0,j}.T(K" - \Delta_{0,i} - \Delta_{0,j}, X - \Delta_0) & \qquad V_j(a) \mid V_j(0) \\
+ \sum_{\substack{1 \le a \le l \\ a \ne j}} \frac{v_{j,a}.(v_{j,a}-1)}{2}.T(K" - \Delta_{0,i} - 2\Delta_{j,a} + \Delta_{a,a}, X - \Delta_0) & \qquad V_j(a) \mid V_j(a) \\
+ \frac{v_{0,j}.(v_{0,j}-1)}{2}.T(K" - \Delta_{0,i} - 2\Delta_{0,j} + \Delta_{j,j}, X - \Delta_0) & \qquad V_j(0) \mid V_j(0) \\
+ \frac{v_{j,j}.(v_{j,j}-2)}{2}.T(K" - \Delta_{0,i} - \Delta_{j,j}, X - \Delta_0) & \qquad V_j(j) \mid V_j(j)
\end{aligned}
\tag{18}
$$

For $k = i$, we consider the two extremities of the (i, i)-path at the same time. Therefore, this situation is similar to the previous one, except that we have to choose one of the extremities in each case. There are twice as many possibilities as in the previous case.

$$
\begin{aligned}
T(K", X) = T(K", X - 2\Delta_i) & \qquad \emptyset \\
+ \sum_{1 \le a \le l} 2v_{j,a}.T(K" - \Delta_{i,i} - \Delta_{j,a} + \Delta_{i,a}, X - 2\Delta_i) & \qquad V_j(a) \\
+ 2v_{0,j}.T(K" - \Delta_{i,i} - \Delta_{0,j} + \Delta_{i,j}, X - 2\Delta_i) & \qquad V_j(0) \\
+ \sum_{1 \le a < b \le l} 2v_{j,a}.v_{j,b}.T(K" - \Delta_{i,i} - \Delta_{j,a} - \Delta_{j,b} + \Delta_{a,b}, X - 2\Delta_i) & \qquad V_j(a) \mid V_j(b) \\
+ \sum_{1 \le a \le l} 2v_{j,a}.v_{0,j}.T(K" - \Delta_{i,i} - \Delta_{0,j}, X - 2\Delta_i) & \qquad V_j(a) \mid V_j(0) \\
+ \sum_{\substack{1 \le a \le l \\ a \ne j}} v_{j,a}.(v_{j,a}-1).T(K" - \Delta_{i,i} - 2\Delta_{j,a} + \Delta_{a,a}, X - 2\Delta_i) & \qquad V_j(a) \mid V_j(a) \\
+ v_{0,j}.(v_{0,j}-1).T(K" - \Delta_{i,i} - 2\Delta_{0,j} + \Delta_{j,j}, X - 2\Delta_i) & \qquad V_j(0) \mid V_j(0) \\
+ v_{j,j}.(v_{j,j}-2).T(K" - \Delta_{i,i} - \Delta_{j,j}, X - 2\Delta_i) & \qquad V_j(j) \mid V_j(j)
\end{aligned}
\tag{19}
$$

Now, if we have $\forall k, x_k = 0$, then all the vertices have been considered and:

$$
T_{i,j}(G_1, K_1, K_2, 0) = \begin{cases} 1 \text{ if } K_1 = K_2 \\ 0 \text{ otherwise} \end{cases} \tag{20}
$$

The table of all possible $T_{i,j}(K_1, K_2, X)$ is of size $l^2.n^{l(l+4)}$. Using the previous equations, we can compute the table by increasing X in $O(n^{l(l+4)})$ operations (individual equations are independent of n). We now have $N_{i,j}(K', K) = T_{i,j}(K', K, X)$ where $\forall k \neq i, x_k = k'_{i,k}$ and $x_i = 2k'_{i,i}$.