Distributed Computation of Ribbon Tableaux and Spin Polynomials

Sébastien Veigneau*

Institut Gaspard Monge, Université de Marne-la-Vallée, 2, rue de la Butte Verte, 93166 Noisy-le-Grand Cedex, France

Abstract. Recent works in algebraic combinatorics have brought up to date the importance of certain planar structures, called ribbon tableaux, which are generalizations of Young tableaux. This paper gives an algorithm to efficiently distribute, using PVM, the computation of the set of all ribbon tableaux of given shape and weight. It also provides a way to compute the spin polynomials associated to those sets of ribbon tableaux, these polynomials leading to generalizations of Hall-Littlewood functions.

1 Introduction

Ribbon tableaux, introduced by Stanton and White [9] can be used to construct a basis of highest weight vectors in the Fock space representation of the quantum affine algebra $U_q(\widehat{sl}_n)$ [7]. This basis plays a crucial rôle in the computation of the canonical basis of the q-deformed Fock space. It is conjectured in [8] that this basis gives the decomposition matrices of q-Schur algebras at roots of unity. Also, generating functions of ribbon tableaux lead to generalizations of Hall-Littlewood functions, which are q-analogues of products of Schur functions [7].

We present a distributed algorithm for generating sets of ribbon tableaux or computing the generating functions according to a statistic called spin. We describe a `PVM` implementation of this algorithm and discuss experimental results.

Basic definitions relative to partitions, Young tableaux and symmetric functions can be found in [3] and [6].

2 Ribbon Tableaux

A ribbon tableau is obtained by filling a Ferrers diagram with *ribbons*. A *k-ribbon* is a connected skew diagram of boxes containing no 2×2 square of boxes. Not all Ferrers diagrams are tilable with k-ribbons. An obvious necessary condition is that the number of boxes should be divisible by k but this is not sufficient. For instance, the shape $\lambda = (3, 2, 1)$ is not tilable with dominoes.

Having filled a shape λ with k-ribbons, we label each ribbon by a positive integer in such a way that labels are weakly increasing across rows from left

* Supported by Procope.

to right and strictly increasing across columns bottom-up. More precisely, let the *root* of a ribbon be its rightmost and lowest box. The exact condition on columns is that the root of a ribbon labeled by i must not be above any box of a ribbon labeled by $j \geq i$. The *weight* of a ribbon tableau is defined as in the case of Young tableaux. Let R be a k-ribbon, $h(R)$ its *height* and $w(R)$ its width. Observe that $h(R) + w(R) = k + 1$. The *spin* of R is $s(R) := \frac{h(R)-1}{2} \in \frac{1}{2}\mathbb{N}$. The *spin* of a tableau T is $s(T) := \sum_R s(R)$ where the sum is over all ribbons of the tableau. For two partitions λ and μ such that $|\lambda| = k|\mu|$, we denote by $\mathrm{Tab}_k(\lambda, \mu)$ the set of all k-ribbon tableaux of shape λ and weight μ. The most important information is the *spin polynomial* $G^{(k)}_{\lambda,\mu}(q) := \sum_T q^{s(T)}$ in which the sum is over all $T \in \mathrm{Tab}_k(\lambda, \mu)$. More details about ribbon tableaux can be found in [3] and [7].

We encode a ribbon tableau $T \in \mathrm{Tab}_k(\lambda, \mu)$ by a matrix $M(T)$ with r rows and s columns, r being the length of the partition μ and s the length of the conjugate partition λ' of λ. This matrix is filled with entries 0 or k, the i-th row containing the value k exactly μ_i times. To obtain the i-th row, we look for the indices of columns containing the roots of ribbons labeled i, write k in these columns and other entries are set to 0. To rebuild T from $M(T)$, we first insert the ribbons labeled by 1. The first row of the matrix $M(T)$ indicates the positions of the roots of the ribbons but we have to "fold up" these ribbons. Assume the matrix has s columns and consider the vector $\rho_s := [s-1, s-2, \ldots, 2, 1, 0]$, then add this vector to the first row of the matrix. Now, sort the resulting vector in decreasing order, subtract ρ_s and obtain the conjugate of the shape occupied by ribbons labeled by 1. A given shape has at most one tiling with ribbons carrying the same label. To build the shape occupied by ribbons labeled by 2, we start with the sorted vector obtained at the previous step and add the second row of the matrix. Then, we proceed exactly as for the first row to obtain the shape occupied by ribbons labeled by 1 and 2. As we already know the shape occupied by ribbons labeled by 1, we recover the shape occupied by ribbons labeled by 2. We repeat this construction for ribbons labeled by 3, 4 and so on.

This decoding allows to compute the spin of a ribbon tableau. Indeed, having added ρ_s to the first row of $M(T)$, we obtain a new vector and we have to save the number of inversions I of the permutation that sorts this vector in decreasing order. This number is equal to $I = \sum_R \big(w(R) - 1\big)$ so that $I = \sum_R \big(w(R) - 1\big) = \sum_R \big(k - h(R)\big)$, which leads to $s(T) = \big(n(k-1) - I\big)/2$ where n is the number of k-ribbons in the partial ribbon tableau T.

3 Recursive Computation of Spin Polynomials

The matrix coding leads to a recursive formula for the polynomials:

$$F^{(k)}_{\alpha,\mu}(q) := \sum_{T \in \mathrm{Tab}_k(\lambda,\mu)} \Big(\sum_{R \in T} q^{(w(R)-1)}\Big) \ , \tag{1}$$

where $\alpha = \lambda' + \rho_s$, with $s \geq \lambda_1$. Let $\mu = (\mu_1, \ldots, \mu_r)$. We obtain the possible positions of the μ_r ribbons labeled by r in a ribbon tableau of shape λ and weight

μ by reversing the previous decoding algorithm: the global shape containing all ribbons labeled by $1, \ldots, r-1$ are obtained by subtracting from α all distinct permutations of the vector $v_r := [0, \ldots, 0, k, \ldots, k]$ (k repeated μ_r times), sorting these vectors in decreasing order and storing inversion numbers. We thus build a tree, rooted by α, with edges indexed by all distinct permutations v_r^σ of v_r and with vertices labeled by the sorted vectors $\alpha - v_r^\sigma$, stored with the number of inversions of the sorting permutation. If a vertex contains a vector with repetitions, it cannot belong to a matrix describing a ribbon tableau, so that all edges leading to such vertices are eliminated. Let $\alpha_1, \ldots, \alpha_m$ denote the remaining vertex labels, and $I_1, \ldots, I_m$ the corresponding inversion numbers. Let also $\mu^* = (\mu_1, \ldots, \mu_{r-1})$. Then, the recurrence formula is:

$$F_{\alpha,\mu}^{(k)}(q) := \sum_{p=1}^{m} q^{I_p} F_{\alpha_p,\mu^*}^{(k)}(q) \ , \qquad (2)$$

with the condition $F_{\rho_s,\emptyset}^{(k)} := 1$. Now, the spin polynomial is given by the formula:

$$G_{\lambda,\mu}^{(k)}(q) = q^{\frac{|\mu|(k-1)}{2}} F_{\alpha,\mu}^{(k)}(q^{-\frac{1}{2}}) \ . \qquad (3)$$

This is actually a polynomial (no fractional exponent) if all parts of λ are divisible by k (the most interesting case).

4 Parallel Implementation with PVM

We now present a parallel implementation of the previous algorithm on a network of computers clustered into a parallel virtual machine described via `PVM`.

We divide the computation into two parts, one being managed by the so-called *master* and the other one by the *slaves*, the master and the slaves being tasks in the virtual machine.

On the one hand, the master has to initiate, administrate and terminate the computation of the spin polynomial. Initiate the computation means that it has to read the input data λ, μ and the maximal number of slaves. It also has to administrate the computation of the spin polynomial, *i.e.* to agree or disagree with the requests of creation of new tasks. Then finally, it has to terminate the computation, that is, to give the spin polynomial $G_{\lambda,\mu}^{(k)}(q)$.

On the other hand, slaves investigate all possibilities. They correspond to the nodes of the previous tree, including the root. Each task has to enumerate all distinct permutations v_r^σ of v_r (labeling the edges) and to eliminate the non admissible ones. Then, it asks the master whether it should create new tasks corresponding to the remaining edges, creates the accepted new tasks with their parameters, informs the master that it is dying and finally dies.

The master reads the input data and creates the root of the tree, *i.e.* spawns the first task. Then, it enters a `while` loop that terminates when there is no more alive task in the virtual machine. The master is thus waiting for any message from any task. It can be a message informing that a task is dying which corresponds either to a successful result at the lowest level in the tree, or simply to a node that has terminated its work. The master also receives messages carrying requests for new task creations. The answer depends upon several parameters. The master has to check whether the maximal number of spawned tasks is already reached or not. If this number is already reached then the master informs the task not to spawn new tasks but to recursively compute its subtree. Furthermore, at a given level of the tree, several sorted vectors may be equal, and the master has to manage the tasks so that a given subtree is computed only once, even if two – or more – tasks have sent to the master the same request. This remains true even if a task is sequentially computing its subtree, requests being then sent before recursive calls. In other words, it is transparent for the master, which only has to count the number of spawned tasks among the total number of tasks.

When all tasks are terminated, the master builds the spin polynomial from the partial results obtained from tasks of the lowest level in the execution graph.

We can for example obtain the spin polynomial for 6-ribbon tableaux of shape $(18, 18, 18, 18, 18, 18, 18, 18)$ and weight $(4, 4, 4, 4, 4, 4)$ (1079706100 elements) with about 7 minutes on a cluster of 10 HP9000 workstations. This computation takes 4 minutes and 30 seconds with 15 computers.

A detailed version of this paper can be found on the Internet at the URL: *http://phalanstere.univ-mlv.fr/~sv*.

References

1. Betten, A., Veigneau, S., Wassermann, A.: Parallel Computation in Combinatorics using PVM, preprint, Université de Marne-la-Vallée, 1996.
2. Carré, C., Leclerc, B.: Splitting the square of a Schur function into its symmetric and antisymmetric parts. J. Alg. Comb. **4** (1995), 201–231.
3. Désarménien, J., Leclerc, B., Thibon J.-Y.: Hall-Littlewood functions and Kostka-Foulkes polynomials in representation theory. Séminaire Lotharingien de Combinatoire, Université de Strasbourg, 1993 (URL: *http://cartan.u-strasbg.fr:80/~slc/opapers/s32leclerc.html*).
4. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: PVM: Parallel Virtual Machine, A users guide and tutorial for networked parallel computing. MIT Press, 1994.
5. Kirillov, A. N., Lascoux, A., Leclerc, B., Thibon, J.-Y.: Séries génératrices pour les tableaux de dominos. C.R. Acad. Sci. Paris **318** (1994), 395–400.
6. Knuth, D.: The art of computer programming. Vol. **3**, Addison-Wesley, 1981.
7. Lascoux, A., Leclerc, B., Thibon J.-Y.: Ribbon tableaux, Hall-Littlewood functions, Quantum affine algebras and unipotent varieties. preprint IGM **96.2**, q-alg/9512031.
8. Leclerc, B., Thibon J.-Y.: Canonical Bases of q-Deformed Fock Spaces. International Math. Research Notices, to appear.
9. Stanton, D., White, D.: A Schensted algorithm for rim-hook tableaux. J. Comb. Theory A **40** (1985), 211–247.

This article was processed using the LaTeX macro package with LLNCS style