

A GENERALIZED INSERTION ALGORITHM FOR RC GRAPHS TO PROVE PIERI'S COLUMN MULTIPLICATION RULE

ABHINAV KUMAR

1. INTRODUCTION

We prove

Theorem 1.1.

$$P_w P_{c[r,m]} = \sum P_{w'}$$

where the sum is over all $w' = wt_{a_1,b_1} \dots t_{a_m,b_m}$ such that the a_i 's are distinct and less than or equal to r , $r < b_1 \leq b_2 \leq \dots \leq b_m$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(a_j) < w'(a_i)$ for every $i < j$ with $b_i = b_j$.

This may also be stated as follows:

Theorem 1.2.

$$P_w P_{c[r,m]} = \sum P_{w'}$$

where the sum is over all $w' = wt_{a_1,b_1} \dots t_{a_m,b_m}$ such that the a_i 's are distinct and less than or equal to r , $r < b_1 \leq b_2 \leq \dots \leq b_m$, satisfying $l(w') = l(w) + i$ for every $i \leq m$.

2. A COLUMN INSERTION ALGORITHM

Note the the rc-graphs of $c[r, m]$ can be completely characterised by a set of distinct numbers $i_1 > i_2 > \dots > i_m$ which are all less than or equal to r . This corresponds to one crossing each in the rows i_1, \dots, i_m .

Algorithm: Given an rc-graph D , and a level r , suppose we have to insert elements into row i_1, \dots, i_m , where $r \geq i_1 > i_2 > \dots > i_m$. We keep an ordered sequence of (a_i, b_i) 's during the course of the algorithm, such that $r < b_1 \leq b_2 \leq \dots \leq b_l$, each $a_i \leq r$ and all a_i 's are distinct. Initially the sequence is empty.

Proceed row by row as follows. Starting with row i_1 , find the rightmost position (i_1, j) where the configuration is one of the following:

$$\begin{array}{ccc}
 \begin{array}{c} s \text{---} \bigcup \\ \bigcap \text{---} \\ q \end{array} & \begin{array}{l} \text{with } s \leq r < q, \\ s \neq a_j \text{ for any } j. \end{array} & \begin{array}{c} s \text{---} \bigcup \\ \bigcap \text{---} \\ a_i \end{array} & \begin{array}{l} \text{with } s \leq r, \\ s \neq a_j \text{ for any } j. \end{array}
 \end{array}$$

Figure 3.

Add this intersection. If we are in the first case, insert (s, q) into the sequence (a_i, b_i) in the rightmost position, such that b_i 's remain nondecreasing in the sequence. If we are in the second case add (s, b_i) just before where (a_i, b_i) is in the sequence. We are now done with row i_1 . However, the graph we have obtained

might not be an rc-graph. So we go up to the next row and perform the following rectification procedure. Starting from the left, we look for either of the two configurations:



Figure 4.

We remove the intersection, delete the pair (a_i, b_i) from the sequence of (a, b) 's. If the intersection was at position (i, j) , find the maximal $j' < j$ such that the configuration at (i, j') is of the form shown in Figure 3. (Such a position must exist. If a (b_i, a_i) intersection is removed, then since the b_i is greater than r , we will be able to find such a position. If (a_j, a_i) intersection is removed, then the a_j strand must go down to the row below, otherwise it would not have been on the list at all. Therefore we must run into the situation shown in Figure 3.) We add the intersection and to the list of (a, b) 's, we add the new pair the same way it was described before. Then we look to the right to see if there is another intersection of the form given in Figure 4. If it exists, remove it together with the appropriate (a_i, b_i) and proceed as before. We call this process rectification of a given row.

When we are done with rectifying the current row, we add an intersection to this row if it is one of the i_f 's in the same manner as for the row i_1 , giving us some new pairs (a_j, b_j) . We then proceed to the next row and repeat the procedure. It is clear from the construction that the a 's remain distinct and the b 's remain ordered. \square

The rest of this section is concerned with proving the following:

Theorem 2.1. *The above algorithm produces an rc-graph.*

Before proving Theorem 2.1, we will need two lemmas and another algorithm.

Lemma 2.2. *After the row ℓ has been rectified the strands a_i, \dots, a_{i+k} with $b = b_i = \dots = b_{i+k}$ pass from the row ℓ to the row $\ell - 1$ to the right of the place where the strand b passes from the row ℓ to the row $\ell - 1$. Moreover, they pass from the row ℓ to the row $\ell - 1$ in the opposite order they appear in the (a, b) sequence. (See Figure 5 for an illustration.)*

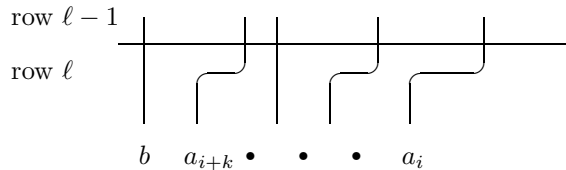


Figure 5. The order in which the strands a_i, \dots, a_{i+k} and b intersect the line, which separates rows ℓ and $\ell - 1$ is the opposite order as that of a_i 's in the (a, b) sequence.

Proof: This lemma can be proved by induction on ℓ . At the beginning of the algorithm, set $\ell = r - 1$. Then the statement of the lemma holds, since the (a, b) sequence is empty.

Assume we know that lemma holds for $\ell + 1$. Notice that if during the rectification of the row ℓ no pairs (a_j, b_j) with $b_j = b$ were added or removed from the (a, b)

sequence, the strands a_i, \dots, a_{i+k} and b never intersect each other in the row ℓ . Hence they pass between the rows ℓ and $\ell - 1$ in the same order they pass from the row $\ell + 1$ to the row ℓ .

Moreover, the order of the strands a_i, \dots, a_{i+k} and b can change only when some of them intersect in the row ℓ . But then this intersection is one of the intersection in Figure 4. The algorithm removes this intersection and it removes the appropriate tuple from the (a, b) sequence to preserve the order of the stands. At the same time, when an intersection is added the addition of (a_j, b_j) to the list is consistent with order of the strands a_i, \dots, a_{i+k} and b passing from the row ℓ to the row $\ell - 1$. \square

Lemma 2.3. *The permutation carried by the strands of the graph at each moment of the algorithm is $wt_{a_1 b_1} \dots t_{a_{m'} b_{m'}}$ where $\langle (a_1, b_1), \dots, (a_{m'}, b_{m'}) \rangle$ is the sequence upto that point in the algorithm.*

Proof: We again use induction: we need to check the above statement remains true when we insert an intersection or delete one.

Assume we are in the first case of Figure 3, that is we are adding an intersection and (s, q) to the list of (a, b) 's. We can easily see that adding this intersection multiplies the permutation of the graph by $t_{s, q}$ on the right. So, if $\langle (a_1, b_1), \dots, (a_{m'}, b_{m'}) \rangle$ is the list before this insertion, the permutation of the graph after the insertion is $wt_{a_1 b_1} \dots t_{a_{m'} b_{m'}} t_{s, q}$. Using the fact that t_{ab} and t_{cd} commute when a, b, c, d are all distinct, we can commute $t_{s, q}$ through $t_{a_1 b_1} \dots t_{a_{m'} b_{m'}}$ to the left until some $b_j \leq q$. Then the permutation carried by our graph is $wt_{a_1 b_1} \dots t_{a_j b_j} t_{s, q} \dots t_{a_{m'} b_{m'}}$. So, after inserting (s, q) into the list after (a_i, b_j) , the property we are trying to prove holds.

If we are in the second situation of Figure 3, then we multiply the permutation $wt_{a_1 b_1} \dots t_{a_{m'} b_{m'}}$ by t_{s, a_i} on the right. Then t_{s, a_i} can be moved to the left until the place i , since all a_j 's are distinct and thus t_{s, a_i} commutes with all t_{a_j, b_j} if $j > i$. After that we use the following identity, which shows why we have to add (s, b_i) to the (a, b) list before (a_i, b_i) :

$$(2.1) \quad t_{a_i, b} t_{s, a_i} = t_{s, b_i} t_{a_i, b}.$$

For the deletion, we use almost identical arguments in both cases of Figure 4, since the deletion of an intersection also multiplies the permutation of the graph by appropriate transposition. The only difference is that in the second case, instead of (2.1) we have to use the following identity:

$$t_{a_i, b} t_{a_j, b} t_{a_i, a_j} = t_{a_j, b}$$

This concludes the proof of the lemma. \square

To give a proof of Theorem 2.1 we introduce another algorithm (call it Algorithm 2) that we perform on the intermediate graph. This algorithm will take the graph, which was rectified upto the row ℓ , and remove some intersections to produce an rc-graph of the original permutation w .

Algorithm 2. We start with the list of (a_i, b_i) 's and starting with the last row ℓ we have finished with, go down one row by one and from right to left in each row. We look for intersections of the form



Figure 6.

Whenever we see these, we remove the intersection and (a_i, b_i) from its place in the list. Note that both statements of Lemma 2.2 and Lemma 2.3 hold at each moment of this algorithm. Ultimately all the (a_i, b_i) will be removed from the list because each strand a_i definitely intersects the strand b_i at each moment of Algorithm 2 by Lemma 2.2, so all of them will get removed and at the end we will get a graph and an empty list of (a, b) 's. \square

The result of Algorithm 2 will be a graph with the permutation w by Lemma 2.3, with exactly $l(w)$ intersections. Therefore it will be an rc-graph for w . We will use this fact to prove that the main algorithm generates an rc-graph.

We come back to the proof of Theorem 2.1. Suppose we have rectified some row ℓ and inserted some elements in it, and we want to show there are no double intersections below and including that row. Because of the inductive hypothesis, this works for the previous row $\ell + 1$ (base case is clear). Hence we need to check that no two strands intersect at the row ℓ and at some lower row after the algorithm has rectified the row ℓ and inserted all the required intersections into it.

We will show that the only double intersection which might be introduced in the row ℓ by insertions in the lower rows will be removed by the algorithm and no new double intersections will be introduced in this row.

Let us show that after rectifying the row $\ell + 1$, and deleting all the intersections from Figure 4 in the row ℓ at once, there are no strands, which intersect twice below the row ℓ . For this apply Algorithm 2 to the graph constructed after rectifying the row $\ell + 1$. Then the resulting graph is an rc-graph D' of the permutation w . Start applying the inverse Algorithm 2 to this rc-graph D' , that is start adding intersections to D' in the order opposite to the order they were removed. It is then not difficult to see that since we are only adding intersections from Figure 6 and at each moment there are no double intersections below row $\ell + 1$, the only double intersections at the row ℓ have to look like the ones shown on Figure 4. This shows that removing intersections from row ℓ we remove all the double intersections below or at this row.

Now let's check that we do not create any double intersections below or at the row ℓ during rectification of this row or during inserting new intersections into it. Clearly the only way we could have done this is if we inserted an intersection in the position as in the second case of Figure 3, so that the strands s and a_i intersected before, in a lower row. (In the first case of Figure 3, the two strands could not have intersected in a lower row, since $s < q$.) So, we shall show that in the first case the strands s and a_i cannot intersect in a lower row, that is $s < a_i$. Proving this will finish the proof of Theorem 2.1.

After the row ℓ is rectified, we again apply Algorithm 2. Notice that this algorithm keeps moving the strand a_i to the right in the row ℓ till the last point, where (a_i, b_i) is removed from the list of (a, b) 's, at which point the strand a_i moves left in the row ℓ , even to the left of its *original* position. For example, in Figure 7, the circled intersections are removed during Algorithm 2. When the first one is

removed, (a_j, b_j) is removed from the list and a_i moves from x to the right, to z , and when the second intersection is removed, (a_i, b_i) is removed and a_i moves to the left of x , to y .

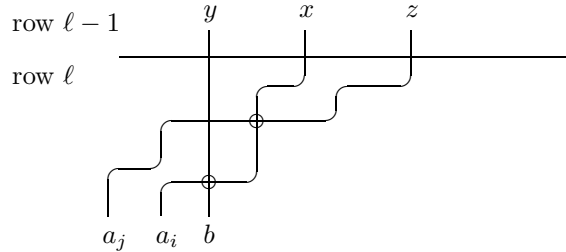


Figure 7. During Algorithm 2, a_i moved from x to z and then to y in the row ℓ .

Therefore the position x of the strand b_i at the row ℓ at the start of Algorithm 2 is to the right of the position y at the row ℓ at the end of it.

Now we look at what happens after (a_i, b_i) is moved off the list in Algorithm 2, by removing an intersection with b_{i_1} (first it moved from x to z and then from z to y in the row ℓ). We look next at the history of b_{i_2} till (a_{i_2}, b_{i_2}) is moved off the list and so on, getting a figure like the following.

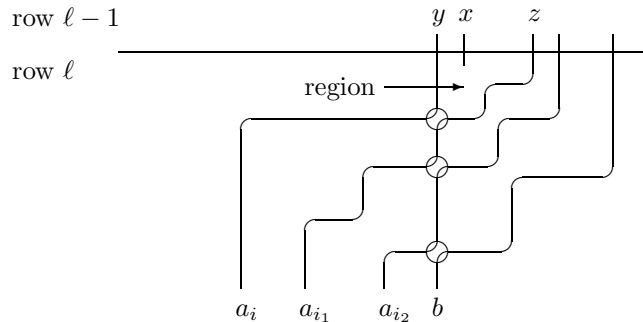


Figure 8: No strands between a_i and b can pass in the shown region

Note that no two consecutive strands shown in the figure can intersect above the encircled spot (where one of them is moved off the list), because that intersection would then have to be removed earlier than the encircled one, by construction of the Algorithm 2. Therefore, from the fact that the outcome of Algorithm 2 is an rc-graph, we conclude that in the region shown, there can pass no strand between a_i and b (otherwise some double intersections must occur). Hence the s from the second case of Figure 3 must always be greater than a_i . This concludes the proof of Theorem 2.1.

3. THE PROOF OF PIERI'S FORMULA.

To prove Pieri's formula using the above insertion algorithm, we will construct an inverse algorithm, which takes an rc-graph for $w' = wt_{a_1, b_1} \dots t_{a_m, b_m}$, such that the a_i 's are distinct and less than or equal to r , $r < b_1 \leq b_2 \leq \dots \leq b_m$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(a_j) < w'(a_i)$ for every $i < j$ such that $b_i = b_j$, and produces an rc-graph for w and an rc-graph for $c[r, n]$ (remember, that rc-graphs of $c[r, m]$ are given by specifying the rows $i_1 > \dots > i_m$ of intersections). Given

the insertion algorithm and its inverse, we will clearly produce a bijection:

$$\mathcal{RC}(w) \times \{i_1, \dots, i_m : r \geq i_1 > i_2 > \dots > i_m\} = \mathcal{RC}(w) \times \mathcal{RC}(c[r, n]) \rightarrow \bigcup \mathcal{R}(w')$$

where w' ranges as stated above.

Below we will produce this inverse algorithm, which will automatically prove the following

Theorem 3.1.

$$P_w P_{c[r, m]} = \sum P_{w'}$$

where the sum is over all $w' = wt_{a_1, b_1} \dots t_{a_m, b_m}$ such that the a_i 's are distinct and less than or equal to r , $r < b_1 \leq b_2 \leq \dots \leq b_m$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(a_j) < w'(a_i)$ for every $i < j$ with $b_i = b_j$.

Inverse Algorithm.

We are given an rc-graph R' for a permutation $w' = wt_{a_1, b_1} \dots t_{a_m, b_m}$ with the distinct a_i 's, which are less than or equal to than r and with $r < b_1 \leq b_2 \dots \leq b_m$, $l(w') = l(w) + m$, and $w'(b) < w'(c) < w'(a)$ for all a, b, c such that (a, b) is some (a_s, b_s) and (c, b) is some (a_t, b_t) with $s < t$. The algorithm is defined as follows: starting from the top (first) row of the rc-graph, we look from right to left for the occurrence of one of the configurations shown below.

$$\begin{array}{c} | \\ a_i - \text{---} \\ | \\ b_i \end{array} \qquad \begin{array}{c} | \\ a_i - \text{---} \\ | \\ a_j \end{array} \quad \text{with } b_i = b_j,$$

Figure 9.

We then remove this intersection from the rc-graph and consequently remove (a_i, b_i) from the list of (a, b) 's. Immediately after we remove such an intersection we look to its right for a configuration of the form

$$\begin{array}{c} | \\ q - \text{---} \\ | \\ s \end{array} \quad \text{with } s \leq r < q, \\ s \neq s_i \text{ for any } i. \qquad \begin{array}{c} | \\ a_i - \text{---} \\ | \\ s \end{array} \quad \text{with } s \leq r, \\ s \neq a_j \text{ for any } j.$$

Figure 10.

If we do not find such a configuration, we say that an intersection is removed from this row and we move on with the algorithm, looking left along that row (and after that, going to the next row) for an intersection of the form shown. (We also record the row number, ad these row numbers will give us an rc-graph for $c[r, m]$.) If we find such a configuration, we add the intersection at this place (i.e. the first configuration of this sort, looking to the right of the removed intersection) and add (q, s) or (s, b_i) to the list of (a, b) 's the same way it was done in the forward insertion algorithm, i.e we add (q, s) to be (a_i, b_i) to preserve the order of b 's ($b_{i-1} \leq b_i < b_{i+1}$) and we add (s, b_i) right after (a_i, b_i) . Then we proceed with the algorithm, looking left for more intersections to remove. We do this till the list of (a, b) 's becomes empty. At the end we are left with a graph for the permutation w . Moreover, the previously recorded row numbers of removed intersections, produce an rc-graph for $c[r, m]$. To see this we must prove, that at most one intersection is actually removed from each

row (that is, not compensated by an added intersection). Consider the case shown in Figure 11 below.

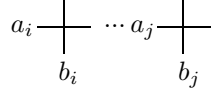


Figure 11

Then after (a_j, b_j) is removed, there must exist a configuration shown in Figure 10 between the uncrossed (a_j, b_j) and the crossed (a_i, b_i) (note that a_j has been struck off the list.) Therefore the uncrossing of (a_j, b_j) will be compensated for.

Now consider the other case shown in Figure 12 below.

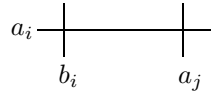


Figure 12

Then after (a_i, a_j) is uncrossed, a_i is no more on the list of a 's so that the (a_i, b_i) will not be uncrossed.

Therefore at most one intersection will be taken off in each row.

The fact that the resulting graph is an rc-graph can be shown using the same sort of technique used for the forward algorithm. It is also evident that the algorithm is the inverse of the forward algorithm. This finishes the proof of Theorem 3.1.

REFERENCES

- [1] N. Bergeron and S. Billey, *RC-graphs and Schubert polynomials*, Experimental Math., **2** (1993), 257-269.
- [2] S. Billey, W. Jockusch, R. Stanley. *Some combinatorial properties of Schubert polynomials*. J. Algebraic Combin. **2** (1993), no. 4, 345–374.
- [3] S. Fomin and A. N. Kirillov, *Yang-Baxter equation, symmetric functions, and Schubert polynomials*, Discrete Mathematics **153**, (1996) 123-143.
- [4] S. Fomin and R. Stanley, *Schubert polynomials and the nil-Coxeter algebra*. Adv. Math. **103** (1994), no. 2, 196–207.
- [5] M. Kogan *RC-graphs and Littlewood-Richardson Rule*. to appear in Internat. Math. Res. Notices.
- [6] A.Lascoux and M. P. Schutzenberger, *Polynomes de Schubert*, C.R.Acad. Sci. Paris **294** (1982) 447-450.
- [7] A.Lascoux and M. P. Schutzenberger, *Schubert polynomials and the Littlewood-Richardson Rule*, Let. Math. Phys. **10** (1985) 111-124.
- [8] I. G. Macdonald, *Notes on Schubert Polynomials*, Publications du LACIM **6**, Universite du Quebec a Montreal (1991)bb
- [9] L. Manivel. *Fonctions symtriques, polynmes de Schubert et lieux de dgnrescence*. Cours Specialiss, 3. Socit Mathmatique de France, Paris, 1998.
- [10] Alexander Postnikov, *On a Quantum Version of Pieri's Formula*, preprint March 23, 1997.
- [11] F. Sottile, *Pieri's formula for flag manifolds and Schubert polynomials*, Annales de l'Institut Fourier **46** (1996), 89-110.
- [12] Fulton, W., *Young tableaux : with applications to representation theory and geometry*, Cambridge University Press, 1997.

- [13] R. Winkel. *A combinatorial bijection between Standard Young Tableaux and reduced words of Grassmannian permutations*. Sem. Loth. Comb. B36h (1996).
- [14] M.Kogan and A.Kumar *A Proof of Pieri's Formula using Generalized Schensted insertion algorithm for RC-Graphs*. to appear in Proceedings of AMS.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MA 02139
E-mail address: `abhinavk@mit.edu`