

A PROOF OF PIERI'S FORMULA USING GENERALIZED SCHENSTED INSERTION ALGORITHM FOR RC-GRAPHS.

MIKHAIL KOGAN AND ABHINAV KUMAR

ABSTRACT. We provide a generalization of the Schensted insertion algorithm for rc-graphs of Bergeron and Billey [1]. The new algorithm is used to give a new proof of Pieri's formula.

1. INTRODUCTION

RC-graphs were first defined by Fomin and Kirillov [3] and later studied by Bergeron and Billey [1]. These are a set of objects that encode the monomials contributing to the expansion of Schubert polynomials. These polynomials were introduced by Lascoux and Schutzenberger [6] [7] and described at length by Macdonald [8] and by Manivel [9].

A central problem in the theory of Schubert polynomials is to provide effective ways of computing the generalized Littlewood-Richardson coefficients c_{vw}^u in the expansion

$$(1.1) \quad P_v P_w = \sum_u c_{vw}^u P_u$$

where P_w is the Schubert polynomial of a permutation w .

The first attempt to compute the generalized Littlewood-Richardson coefficients using rc-graphs was made in [1], where Monk's formula was proved using a generalized Schensted insertion algorithm. Later, in [5], this generalized algorithm was used to compute a more general set of Littlewood-Richardson coefficients. Unfortunately, this algorithm does not work in general and, in particular, fails to give a proof of Pieri's formula. In this paper we will modify this algorithm to inserting a whole row of elements at once instead of an element by element insertion. This generalization will prove Pieri's formula.

Let us give a statement of Pieri's formula, which was originally proved in [6] and later reproved by other methods [11], [10], [13]. Our way of stating Pieri's formula is not standard, but in Section 4 we show how to deduce the standard formulation of Pieri's formula from Theorem 4.1.

Theorem 4.1.

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

where the sum is over all $w' = wt_{a_1, b_1} \dots t_{a_m, b_m}$ such that the b_i 's are distinct and greater than r , $a_1 \leq a_2 \leq \dots \leq a_m \leq r$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(b_j) < w'(a_i)$ for every $i < j$ with $a_i = a_j$.

In the above statement, $\sigma[r, m]$ stands for the permutation $[1, 2, \dots, r-1, r+m, r, r+1, \dots]$. Note that $P_{\sigma[r,m]}$ is equal to the homogeneous symmetric polynomial $h_m(x_1, \dots, x_r)$.

We will define Schubert polynomials, rc-graphs and point out their relationship in Section 2. The generalized insertion algorithm will be given in Section 3, while in Section 4 we will prove Pieri's formula by providing the inverse algorithm.

2. RC-GRAPHS

Let us decide on some notation and state some basic facts about permutations and Schubert polynomials first. For a permutation $w \in S_n$, the symmetric group on n elements, we can write w as $[w(1), \dots, w(n)]$. We let t_{ab} denote the transposition which takes a to b and vice versa, leaving other elements fixed, and write s_i for $t_{i,i+1}$. It can be shown that s_1, \dots, s_{n-1} generate S_n with the relations

$$\begin{aligned} s_i^2 &= 1, \\ s_i s_j &= s_j s_i \text{ if } |i - j| \geq 2, \\ s_i s_{i+1} s_i &= s_{i+1} s_i s_{i+1}. \end{aligned}$$

For a permutation $w \in S_n$, we let the length $l(w)$ of w be the number of inversions of w , i.e. the number of pairs (i, j) with $i < j$ and $w(i) > w(j)$. A string $a_1 a_2 \dots a_p$ such that $s_{a_1} \dots s_{a_p} = w$ with p minimal is called a reduced word for w . It can be shown that this p is just the length of w , and any two reduced word decompositions can be transformed into each other using the last two relations given above. Denote by $R(w)$ the set of all the reduced words for w .

This enables us to define Schubert polynomials. Let ∂_i be an operator which acting on a polynomial $f(x_1, \dots, x_n)$ to the left, yields the polynomial

$$\frac{f(x_1, \dots, x_i, x_{i+1}, \dots, x_n) - f(x_1, \dots, x_{i+1}, x_i, \dots, x_n)}{x_i - x_{i+1}}$$

It can be verified that

$$\begin{aligned} \partial_i^2 &= 0, \\ \partial_i \partial_j &= \partial_j \partial_i \text{ if } |i - j| \geq 2 \\ \partial_i \partial_{i+1} \partial_i &= \partial_{i+1} \partial_i \partial_{i+1} \end{aligned}$$

Thus for any permutation w , we may define ∂_w as $\partial_{a_1} \dots \partial_{a_p}$, where $a_1 \dots a_p$ is *any* reduced word for w . Now the Schubert polynomial P_w is defined to be

$$P_w = \partial_{w^{-1}w_0} x_1^{n-1} x_2^{n-2} \dots x_{n-1}^1 x_n^0$$

where $w_0 = [n, \dots, 1]$ is the longest permutation in S_n .

Now we come to rc-graphs. Given a reduced word $a_1 \dots a_p$ for w , we say that a sequence $\alpha_1, \dots, \alpha_p$ is compatible for the word if

$$\begin{aligned} \alpha_1 &\leq \alpha_2 \leq \dots \leq \alpha_p \\ \alpha_i &\leq a_i, \forall i \\ \alpha_i &< \alpha_{i+1} \text{ if } a_i < a_{i+1} \end{aligned}$$

Denote by $C(\mathbf{a})$ the set of all compatible sequences for $\mathbf{a} \in R(w)$.

Given such a compatible sequence we make an rc-graph by placing intersections (see Figure 1) at all the positions $(\alpha_k, a_k - \alpha_k + 1)$, these form a subset of $\{1, 2, \dots\} \times \{1, 2, \dots\}$. Now we draw strands starting at each row and winding their way up. Wherever there is an intersection symbol, we make the two strands there intersect. It is easy to show by induction that the strand starting at the i^{th} row ends up at the $w(i)^{\text{th}}$ column and that no two strands intersect twice. Conversely, a graph with intersections, such that no two strands intersect twice, immediately gives us a reduced word for the permutation and also a compatible sequence (we look at

the intersections in order of increasing row number, going from right to left in each row, the row number gives the α_i and to get a_i we add the row number and column number and subtract 1.) For more details see [1].

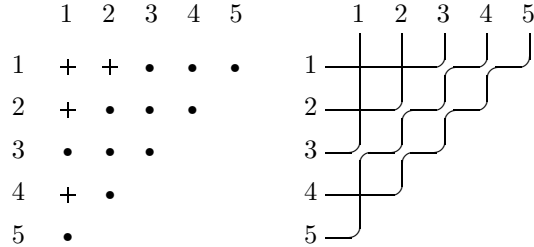


Figure 1: Two ways of drawing an rc-graph.

Denote by $\mathcal{RC}(w)$ the set of all rc-graphs, which correspond to w . For $D \in \mathcal{RC}(w)$ let $x^D = \prod_{(\alpha_k, a_k - \alpha_k + 1) \in D} x_{\alpha_k}$. Now we state a theorem proved in [2] and in [4].

Theorem 2.1. *For any permutation $w \in S_\infty$,*

$$P_w = \sum_{\mathbf{a} \in R(w)} \sum_{\alpha_1 \dots \alpha_p \in C(\mathbf{a})} x_{\alpha_1} \dots x_{\alpha_p} = \sum_{D \in \mathcal{RC}(w)} x^D$$

On an rc-graph we can define a ladder move (of size m) as shown in the Figure 2. Note that after the move we get an rc-graph corresponding to the same permutation (since the number of intersections is preserved and the same strands intersect in the two figures).

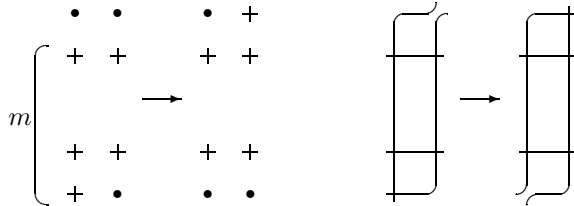


Figure 2: Ladder moves

We shall call the inverse operation an inverse ladder move. Billey and Bergeron in [1] prove that all graphs in $\mathcal{RC}(w)$ can be obtained by applying a succession of ladder moves to $D_{bot}(w) = \{(i, c) : c \leq m_i\}$ (where $m_i = \#\{j : j > i \text{ and } w(j) > w(i)\}$), which is the “bottom” rc-graph corresponding to w . Note that the bottom graph has the property that the rows are left-justified. For instance, the graph shown in Figure 1 is a bottom graph.

3. GENERALIZING INSERTION ALGORITHM.

Recall that Pieri’s rule seeks to give a formula for multiplication of a Schubert polynomial with $P_{\sigma[r,m]}$ (a homogeneous symmetric polynomial). In other words, it computes the generalized Littlewood-Richardson coefficients c_{vw}^u for $w = \sigma[r, m]$,

where the coefficients are define by

$$P_v P_w = \sum_u c_{vw}^u P_u.$$

(We have used the fact that Schubert polynomials form a basis of all polynomials to get a unique such expansion.) To enable us to find this formula, we need a modification of the insertion algorithm, which was defined in [1] and used to compute a family of Littlewood-Richardson coefficients in [5]

Notice that each rc-graph for the permutation $\sigma[r, m]$ can be given by specifying the number of intersections in the rows on or above the row r . Assume this rc-graph is given by k_s elements in the row i_s , where $r \geq i_1 > i_2 > \dots > i_t > 0$ and $\sum k_s = m$. Below is the algorithm which inserts this rc-graph into a general rc-graph.

Algorithm: Given an rc-graph D , and a level r , suppose we have to insert k_s elements into row i_s of D , where $r \geq i_1 > i_2 > \dots > i_t$. We keep an ordered sequence of (a_i, b_i) 's during the course of the algorithm, such that $a_1 \leq a_2 \leq \dots \leq a_l \leq r$, each $b_i > r$ and all b_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:

$$\begin{array}{cc} \begin{array}{c} \text{---} \\ \diagup \quad \diagdown \\ \text{---} \\ | \\ \text{---} \\ q \end{array} & \begin{array}{l} \text{with } s \leq r < q, \\ q \neq b_j \text{ for any } j. \end{array} & \begin{array}{c} \text{---} \\ \diagdown \quad \diagup \\ \text{---} \\ | \\ \text{---} \\ q \end{array} & \begin{array}{l} \text{with } r < q, \\ q \neq b_j \text{ for any } j. \end{array} \end{array}$$

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 a_i 's remain nondecreasing in the sequence. If we are in the second case add (a_i, q) just before where (a_i, b_i) is in the sequence. Repeat the above process k_1 times to insert k_1 intersections into the row i_1 to get a new graph together with a sequence $(a_1, b_1), \dots, (a_{k_1}, b_{k_1})$ where we know for sure that the b_i 's are distinct and greater than r and a_i 's are ordered and less than or equal to r . Also, 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:

$$\begin{array}{cc} \begin{array}{c} | \\ \text{---} \\ | \\ \text{---} \\ a_i \end{array} & \begin{array}{c} | \\ \text{---} \\ | \\ \text{---} \\ b_j \end{array} & \text{with } a_i = a_j, \end{array}$$

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 since the b_i is greater than r and (a_i, b_i) is no longer in the sequence.) 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 what intersections we must to this row in the same manner as for the previous row, 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 b 's remain distinct and the a 's remain ordered. \square

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

Theorem 3.1. *The above algorithm produces an rc-graph.*

Before proving Theorem 3.1, we will need two lemmas and another algorithm.

Lemma 3.2. *After the row ℓ has been rectified the strands b_i, \dots, b_{i+k} with $a = a_i = \dots = a_{i+k}$ pass from the row ℓ to the row $\ell - 1$ to the left of the place where the strand a passes from the row ℓ to the row $\ell - 1$. Moreover, they pass from the row ℓ to the row $\ell - 1$ in the same order they appear in the (a, b) sequence. (See Figure 5 for an illustration.)*

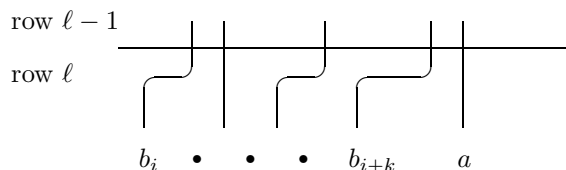


Figure 5. The order in which the strands b_i, \dots, b_{i+k} and a intersect the line, which separates rows ℓ and $\ell - 1$ is the same as the order of b_i 's in the (a, b) sequence.

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 $a_j = a$ were added or removed from the (a, b) sequence, the strands b_i, \dots, b_{i+k} and a 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 b_i, \dots, b_{i+k} and a 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 b_i, \dots, b_{i+k} and a passing from the row ℓ to the row $\ell - 1$. \square

Lemma 3.3. *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 $wt_{a_1b_1}\dots t_{a_{m'}b_{m'}}t_{s,q}$. 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 $t_{a_1b_1}\dots t_{a_{m'}b_{m'}}$ to the left until some $a_j \leq s$. Then the permutation carried by our graph is $wt_{a_1b_1}\dots t_{a_jb_j}t_{sq}\dots t_{a_{m'}b_{m'}}$. 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 $wt_{a_1b_1}\dots t_{a_{m'}b_{m'}}$ by $t_{b_i,q}$ on the right. Then $t_{b_i,q}$ can be moved to the left until the place i , since all b_j 's are distinct and thus $t_{b_i,q}$ 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 (a_i, q) to the (a, b) list before (a_i, b_i) :

$$(3.1) \quad t_{a,b_i}t_{b_i,q} = t_{a,q}t_{a,b_i}.$$

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 (3.1) we have to use the following identity:

$$t_{a,b_i}t_{a,b_j}t_{b_i,b_j} = t_{a,b_j}$$

This concludes the proof of the lemma. \square

To give a proof of Theorem 3.1 we have to 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

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

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 3.2 and Lemma 3.3 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 3.2, 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 3.3, 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 3.1. 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 b_i and q 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 b_i and q cannot intersect in a lower row, that is $b_i < q$. Proving this will finish the proof of Theorem 3.1.

After the row ℓ is rectified, we again apply Algorithm 2. Notice that this algorithm keeps moving the strand b_i to the left 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 b_i moves right in the row ℓ , even to the right 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 b_i moves from x to the left, to z , and when the second intersection is removed, (a_i, b_i) is removed and b_i moves to the right of x , to y .

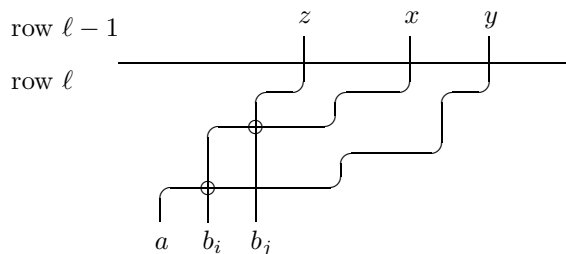


Figure 7. During Algorithm 2, b_i moved from x to z and then to y in the row ℓ .

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

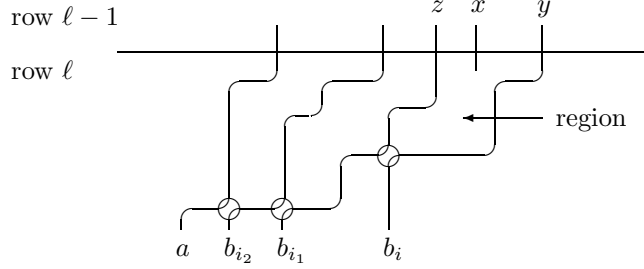


Figure 8. No strands between a and b_i can pass in the shown region.

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 and b_i (otherwise some double intersections must occur). Hence the q from the second case of Figure 3 must always be greater than b_i . This concludes the proof of Theorem 3.1.

Remark 3.4. Assume that the permutation w satisfies the following condition $w(i+1) > w(i)$ if $i > r$. Each rc-graph of such a permutation has the following defining property: no two strand, which start below row r intersect. In this case it can be shown that the above algorithm is equivalent to one by one insertion. This is consistent with the results in [5], where the Pieri formula in this special case was proved using the one by one insertion. The above statement and the fact that one by one insertion does not work in general indicate that this algorithm provides the proper generalization of Bergeron-Billey algorithm of [1].

4. 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 b_i 's are distinct and greater than r , $a_1 \leq a_2 \leq \dots \leq a_m \leq r$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(b_j) < w'(a_i)$ for every $i < j$ such that $a_i = a_j$, and produces an rc-graph for w and an rc-graph for $\sigma[r, n]$ (remember, that rc-graphs of $\sigma[r, m]$ are given by specifying the number of intersections k_s in in each row $s \leq r$, such that $\sum k_s = m$). Given the insertion algorithm and its inverse, we will clearly produce a bijection:

$$\mathcal{RC}(w) \times \{(k_1, \dots, k_r) : \sum k_i = m\} = \mathcal{RC}(w) \times \mathcal{RC}(\sigma[r, n]) \rightarrow \bigcup \mathcal{RC}(w')$$

where w' ranges as stated above.

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

Theorem 4.1.

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

where the sum is over all $w' = wt_{a_1, b_1} \dots t_{a_m, b_m}$ such that the b_i 's are distinct and greater than r , $a_1 \leq a_2 \leq \dots \leq a_m \leq r$, satisfying $l(w') = l(w) + m$ and $w'(b_i) < w'(b_j) < w'(a_i)$ for every $i < j$ with $a_i = a_j$.

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 b_i 's, which are greater than r and with $a_1 \leq a_2 \dots \leq a_m \leq r$, $l(w') = l(w) + m$, and $w'(a) > w'(b) > w'(c)$ for all a, b, c such that (a, b) is some (a_s, b_s) and (a, c) 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.



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

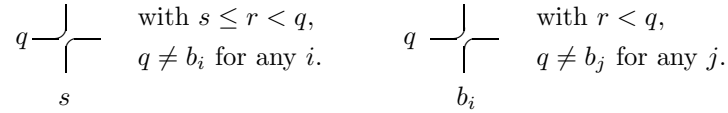


Figure 10.

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. (At the same time we record the number of removed intersections from this row, as these numbers will give an rc-graph for $\sigma[r, m]$ at the end of the algorithm.) 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 (a_i, q) 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 a 's ($a_{i-1} \leq a_i < a_{i+1}$) and we add (a_i, q) right after (a_j, b_j) . 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 numbers of removed intersections in each row, where intersections were removed from the rc-graph, produce an rc-graph for $\sigma[r, m]$.

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 4.1.

Let us now restate Pieri's formula in its original form, as it appeared in [6]. Assume that for the (a, b) sequence from Theorem 4.1 we have $a_1 = \dots = a_{i_1} < a_{i_1+1} = \dots = a_{i_2} < \dots < a_{i_p} = \dots = a_m$. Then $w' = w\zeta_1 \dots \zeta_p$ where each $\zeta_k = (a_{i_{k-1}+1}, b_{i_{k-1}} + 1) \dots (a_{i_k}, b_{i_k})$ is a cycle. Additionally, it follows from Theorem 4.1 that for each ζ_k , there exists exactly one a with $w(a) < w\zeta_k(a)$ and exactly one $a' \leq r$ with $w(a') \neq w\zeta_k(a')$. (Actually, $a = a' = a_{i_k}$.) Hence it is easy to conclude that Theorem 4.1 is equivalent to the following formulation of Pieri's formula given in [6].

Theorem 4.2.

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

where the sum is over all $w' = w\zeta_1 \dots \zeta_p$ satisfying the following conditions: $l(w') = l(w) + m$; each ζ_k is a cycle, such that there exists exactly one a with $w(a) < w\zeta_k(a)$ and exactly one $a' \leq r$ with $w(a') \neq w\zeta_k(a')$.

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 Mathematique 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).

NORTHEASTERN UNIVERSITY, BOSTON, MA 02115

E-mail address: `misha@research.neu.edu`

MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MA 02139

E-mail address: `abhinavk@mit.edu`