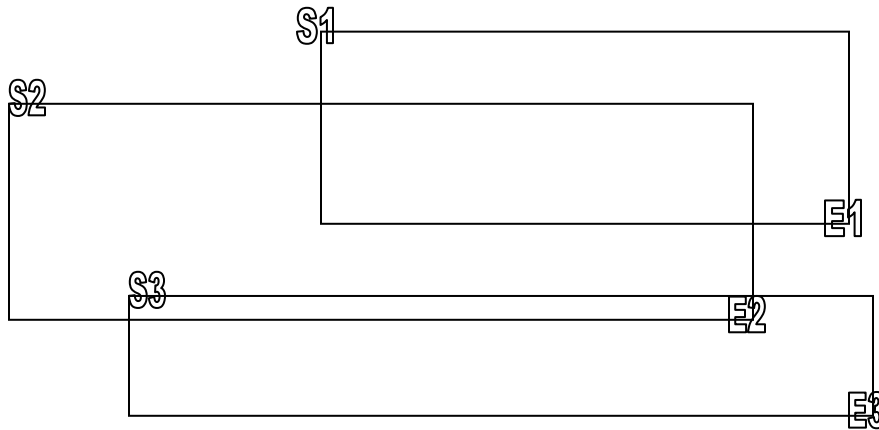


Proof of Gessel-Viennot Algorithm for General Case:

Definition: “Touch”: The **first instance** where two paths have met. Further instances do not count as a touch.

We have seen in previous progress reports that this works for $n = 2$, and speculation that due to the inclusion-exclusion principle it should work in the general case. First let me examine what is being “included” and “excluded”, and examine the $n = 3$ case:

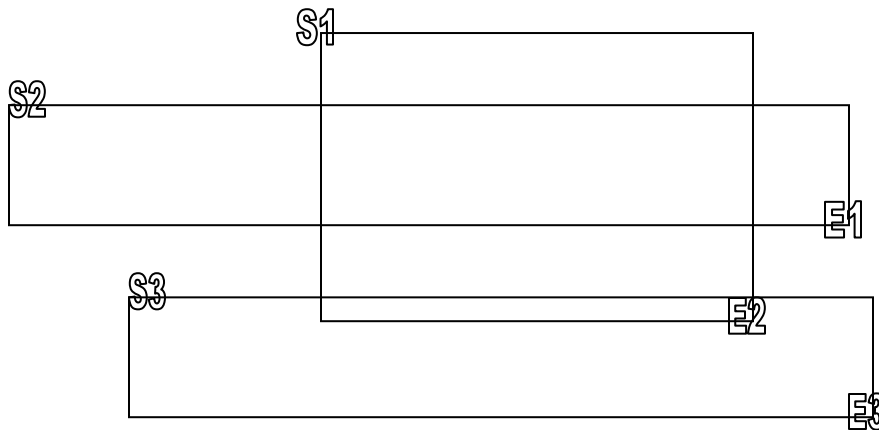
The solution is as follows: what we want is all of the paths that have touched zero times. Multiplying the number of possible paths between all starting and ending points gives the number of paths that have crossed any number of times:



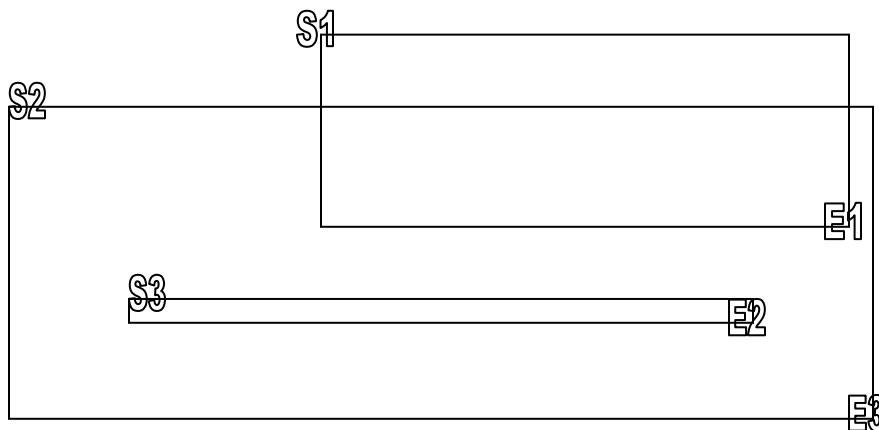
Above, a box denotes the range of possible paths that are covered starting at a starting point and ending at the ending point, e.g. “routings”. Denote this routing (123), where this means that the path starting at 1 ended at 1, 2 ended at 2, and 3 ended at 3.

However, I have now included all paths that have touched another path and I only want those that have touched zero times. For the next step, I will include paths that have touched at least one other path. Pictorially, to get these I select routings by swapping a pair of endpoints: (213), (132), (321)

(213)



(132)



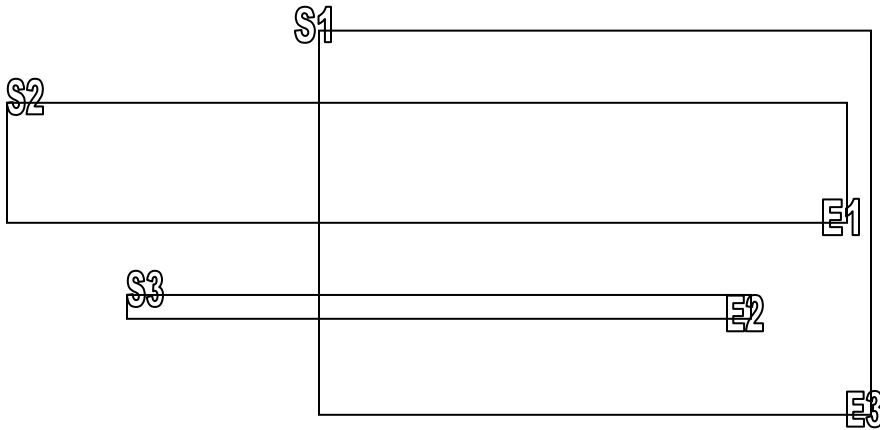
(321) has no valid routings: E1 is above and to the right of E3. Note that here, I may have excluded accidentally some choices that have not, in fact, crossed once. (In paths where this occurs, G-V may fail. See below for details).

Note that with a better choice of initial conditions now that I have included paths that have touched no times, excluded paths that have touched at least once, but now I may have multiply excluded paths that touched twice or more!

Finally, I want to include paths that have touched at least two others, since now these may have been excluded multiple times (all included at first, then excluded maybe more than once after the full set of one-crossing paths, and now I need to add them once more for a total of zero). These come from two swaps: e.g., (231) and (312).

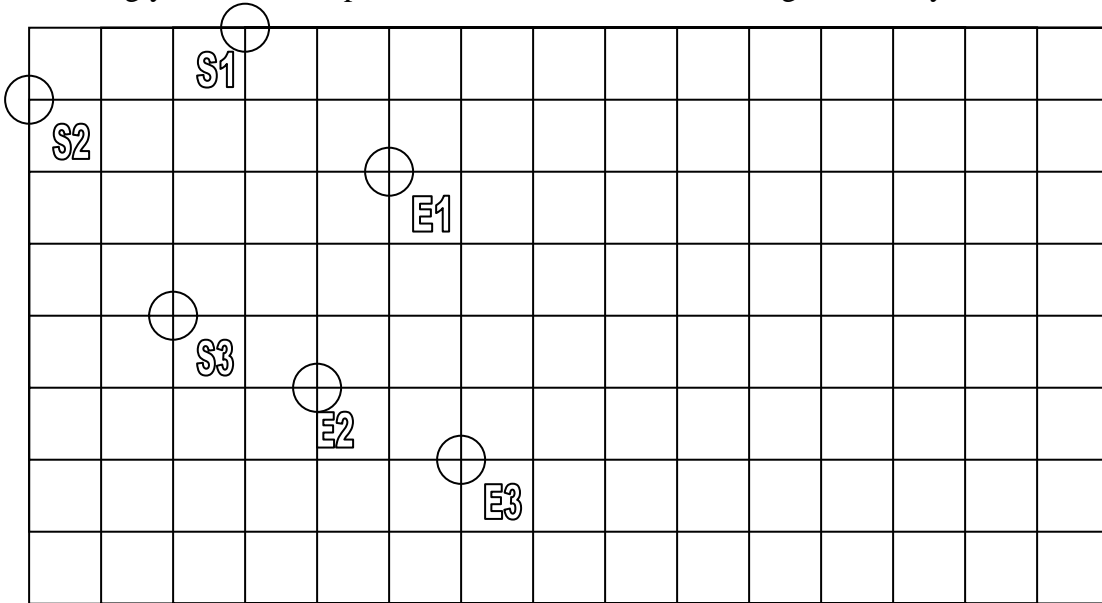
(231) (No routes exist from S3 to E1)

(312)



Note again that while many paths touching twice have been included, a few of these will have no crosses at all, or possibly just one. (this is if E3 was moved right: in fact, this choice of points represents a failure of the algorithm, described below!)

Interestingly, herein lies a point where the Gessel-Viennot algorithm may fail:



The determinant of the matrix given by the algorithm, e.g.:

start pts : (3,0), (0,1), (2,4)

end pts : (5,2), (4,5), (6,6)

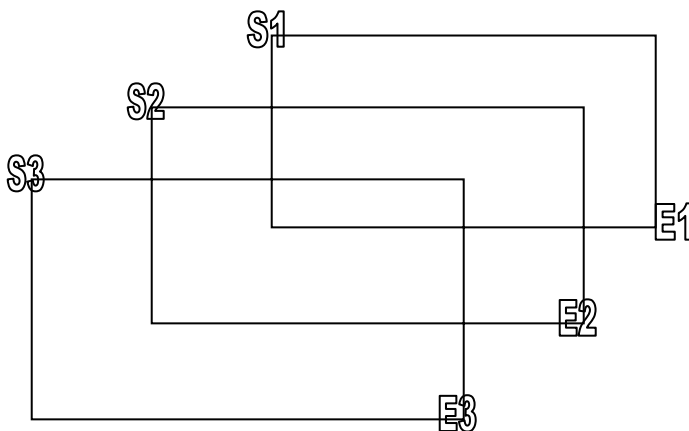
$$\det \begin{bmatrix} \begin{pmatrix} 4 \\ 2 \\ 6 \\ 5 \\ 1 \\ 3 \end{pmatrix} & \begin{pmatrix} 6 \\ 1 \\ 8 \\ 4 \\ 3 \\ 2 \end{pmatrix} & \begin{pmatrix} 9 \\ 3 \\ 11 \\ 6 \\ 6 \\ 4 \end{pmatrix} \end{bmatrix} = \det \begin{bmatrix} 6 & 6 & 84 \\ 6 & 70 & 462 \\ 0 & 3 & 15 \end{bmatrix} = -1044$$

So we see that Gessel-Viennot fails whenever it is possible for two paths to swap endpoints without guaranteeing that the paths themselves cross, otherwise “leftovers” appear and confound the algorithm! In this case, then, I’ll have to use a modification on my original algorithm (described in Jan 15 report, an analog of which is implemented but not in a form applicable to this variety of situation in the basic path counter).

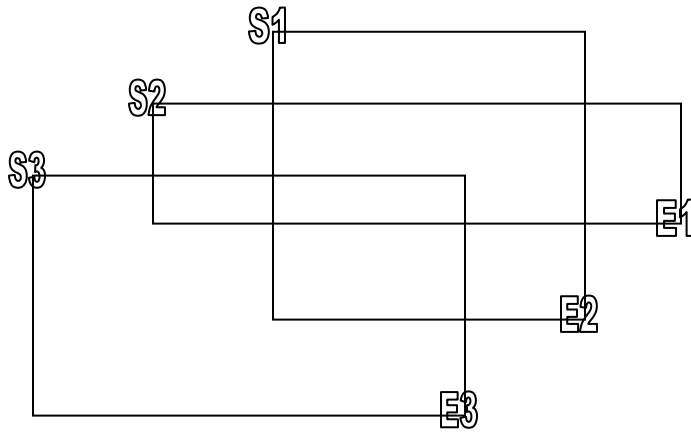
In the case where these leftovers don’t occur, the Gessel-Viennot algorithm works, and it is clear from this argument that its extension may go as such: Suppose that a number of starting and ending points are defined with the stipulation that if any pair of endpoints was swapped, then the corresponding paths would necessarily also cross (example of a case where this criterion would fail above). Now, start by including the case where zero or more are crossing, the “routing” $+(1234)$, for example. Next, exclude all cases where one touch has occurred by subtracting routings where two have been swapped: $-(2134)$, $-(3214)$, $-(4231)$, $-(1324)$, $-(1243)$. Next, I must include all cases where two touches on unique paths have occurred since in the last set of swaps I excluded them possibly multiple times, and then exclude all where three touches have occurred since in the last set of swaps I excluded them all a total of twice over.

Consider how we can be certain that adding the 2-swap permutations must include all paths of 2 touches or more: each 2-touch permutation started as a 1-touch permutation. There are two orders in which this 1-touch could have occurred. Now I see that each one excluded this particular 2-touch permutation once, so that now I need to include it once more to bring my net count of 2-touch permutations to zero.

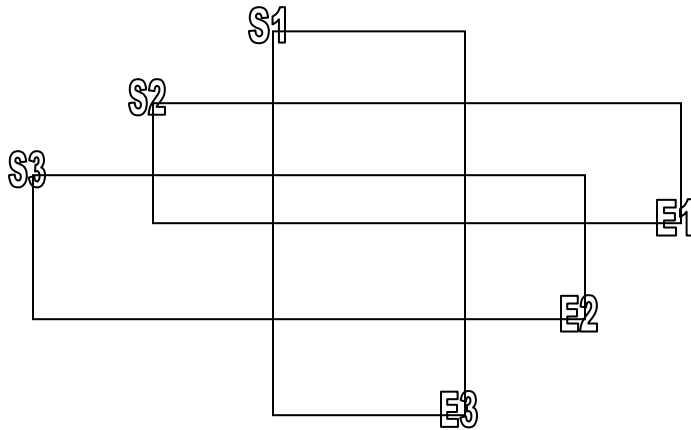
Let me draw this for the 3 case, in a situation where it works:



(123): 0-minimum touches



(213): 1 touch required, possible multi-counts (analogous for the other permutations formable with one swap).



(312): 2 touches required. This is the maximum number of touches on other paths (as usual, only the first touch counts.) There is one other 2-touch permutation which is analogous.