Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 268-279

Dynamic matrix rank with partial lookahead

Telikepalli Kavitha
Indian Institute of Science, Bangalore, India
kavitha@csa.iisc.ernet.in

ABSTRACT. We consider the problem of maintaining information about the rank of a matrix M under changes to its entries. For an $n \times n$ matrix M, we show an amortized upper bound of $O(n^{\omega-1})$ arithmetic operations per change for this problem, where $\omega < 2.376$ is the exponent for matrix multiplication, under the assumption that there is a *lookahead* of up to $\Theta(n)$ locations. That is, we know up to the next $\Theta(n)$ locations $(i_1, j_1), (i_2, j_2), \ldots$, whose entries are going to change, in advance; however we do not know the new entries in these locations in advance. We get the new entries in these locations in a dynamic manner.

1 Introduction

The dynamic matrix rank problem is that of computing the rank of an $n \times n$ matrix $M = \{m_{ij}\}$ under changes to the entries of M. The rank of a matrix M is the maximum number of linearly independent rows (or equivalently, columns) in M. The entries of M come from a field F, and the operation $change_{ij}(v)$ changes the value of the (i, j)-th entry of M to v, where $i, j \in \{1, \ldots, n\}$ and $v \in F$. We have a sequence of $change_{ij}(v)$ operations and the dynamic matrix rank problem is that of designing an efficient algorithm to return the rank of M under every *change* operation.

Here we consider a simpler variant of the above problem, where we assume that we can *lookahead* up to $\Theta(n)$ operations in advance so that we know location indices (i, j) of the entries of M that the next $\Theta(n)$ operations $change_{ij}$ are going to change. Note that we get to know the new value v of m_{ij} only when the operation $change_{ij}(v)$ actually happens, the assumption of lookahead is only regarding the location indices.

1.1 Earlier Work

The dynamic matrix rank problem was first studied by Frandsen and Frandsen [2] in 2006. They showed an upper bound of $O(n^{1.575})$ and a lower bound of $\Omega(n)$ for this problem (the lower bound is valid for algebraically closed fields). Frandsen and Frandsen present two algorithms for the dynamic matrix rank problem - the first algorithm is quite elementary and finds the rank by recomputing a reduced row echelon form of M for every change. This takes $O(n^2)$ time per change. This bound is valid also when a change alters arbitrarily many entries in a single column of the matrix. The second algorithm uses an implicit representation of the reduced row echelon form and this implicit representation is kept sufficiently compact by using fast rectangular matrix multiplication for global rebuilding. This yields a complexity of $O(n^{1.575})$ arithmetic operations per change, and this bound is valid when a change alters up to $O(n^{0.575})$ entries in a single column of M.

Sankowski [7] in 2004 gave several dynamic algorithms for computing matrix inverse, matrix determinant and solving systems of linear equations. The best of these algorithms obtains a worst case time of $O(n^{1.495})$ per change/query. These algorithms assume that the matrix M remains non-singular during the changes. In 2007 Sankowski [8] showed a randomized (there is a small probability of error here) reduction from the dynamic matrix rank problem to the dynamic matrix inverse problem: this yields a randomized upper bound of $O(n^{1.495})$ for the dynamic matrix rank problem.

Dynamic problems with lookahead. Dynamic graph problems with lookahead were considered by Khanna et al. in [5]. However their results have been superseded by dynamic algorithms without lookahead. Very recently, Sankowski and Mucha [9] worked on the dynamic transitive closure problem with lookahead. They present a randomized one-sided error algorithm with changes and queries in $O(n^{\omega(1,1,\epsilon)-\epsilon})$ time given a lookahead of n^{ϵ} operations, where $\omega(1,1,\epsilon)$ is the exponent of multiplication of an $n \times n$ matrix by an $n \times n^{\epsilon}$ matrix. For $\epsilon \leq 0.294$, this yields an algorithm with queries and changes in $O(n^{2-\epsilon})$ time, whereas for $\epsilon = 1$, the time is $O(n^{\omega-1})$, where $\omega < 2.376$ is the exponent for matrix multiplication. Their algorithm is based on a dynamic algorithm with lookahead for matrix inverse. This algorithm also assumes that the matrix M remains non-singular during the changes, which need not be true for the dynamic matrix rank problem. However using the randomized reduction of Sankowski [8] mentioned above, this algorithm for dynamic matrix inverse implies a Monte Carlo algorithm with the same bounds for the dynamic matrix rank problem with lookahead.

In this paper we use a direct approach for solving the dynamic matrix rank problem rather than routing through the dynamic matrix inverse problem. The dynamic matrix rank problem was also originally motivated by its application to the maximum rank matrix completion problem. The maximum rank matrix completion problem is that of assigning values to the undefined entries in a mixed matrix (this is a matrix where some entries are undefined) such that the rank of the resulting fully defined matrix is maximized. Geelen [3] gave a simple $O(n^9)$ time algorithm for the maximum rank completion problem that uses a data structure for dynamic matrix rank. However this application has been superseded by newer results: Berdan [1] reduced this complexity to $O(n^4)$ and Harvey et al. [4] gave an $O(n^3 \log n)$ algorithm for the maximum rank completion problem using a different technique.

Here we show a *deterministic* upper bound of $O(n^{\omega-1})$ for the dynamic matrix rank problem assuming that we are given a lookahead of $O(n)$ location indices. The trade-off between the number of locations that we can lookahead and the running time of our algorithm is: if we are allowed a lookahead of $s \leq n$ locations, then our amortized time per change is $O(n^{\omega}/s + ns^{\omega-2})$. Taking $s = \Theta(n)$ balances the two terms. Our algorithm relies on the idea of maintaining some "important entries" of certain matrices related to M. We describe this in more detail in the next section.

Organization of the paper. We discuss preliminaries in Section 2. Our algorithm for dynamic matrix rank is presented in Section 3. Our update subroutine is described and analyzed in Section 4. Due to lack of space, we omit some proofs from this version of the paper.

2 Preliminaries

We are given an $n \times n$ matrix M with entries from a field F. Our problem is that of computing the rank of M as elements of M change under the $change()$ operations.

As a preprocessing step, in $O(n^\omega)$ time, where $\omega < 2.376$, we compute matrices U and E such that $UM = E$ where E has a special form, similar to reduced row-echelon form. Every row in E that is not the all-zeros row has a special element, whose value is non-zero; let us call this element the "leading element" of its row - such an element is the unique non-zero element in its column. We call such columns *clean* columns (a clean column is a vector with exactly one non-zero coordinate, which is the leading element of its row) and the remaining columns of E are the *dirty* columns. In E we have $rank(M)$ many clean columns, $n - rank(M)$ many dirty columns, $n - rank(M)$ many rows that are all-zeros and $rank(M)$ many non-zero rows. As an example, let us consider the matrix M_0 below to be our starting matrix. To the right we have $U_0M_0 = E_0$ where U_0 is our transformation matrix and E_0 is in our special form. In the preprocessing step we compute U_0 and E_0.

$$M_0 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 5 \end{bmatrix}; \quad \begin{bmatrix} 0 & 0 & 0 & 0.5 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0.5 \\ 2 & 0 & -4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1.5 \\ 0 & 0 & 6 & 9 \end{bmatrix}$$

In the matrix E_0 above, the first, second, and third columns are clean, while the fourth is a dirty column. Let us use the following terminology - right after the preprocessing step, we will define a function $\pi_0 : [n] \to \{0, 1, 2, \ldots, n\}$ that tells us which coordinate of a clean column of E_0 contains the unique non-zero element of that column. That is, only the $\pi_0(i)$-th coordinate of a clean column i is non-zero. So in the example above, we have $\pi_0(1) = 1$, $\pi_0(2) = 3$, and $\pi_0(3) = 4$. We define $\pi_0(i) = 0$ if column i is a dirty column. We will also use the function $\mathsf{Zero}_0 : [n] \to \{true, false\}$ where $\mathsf{Zero}_0(i) = true$ if and only if row i of E_0 is an all-zeros row. In our example, $\mathsf{Zero}_0(2) = true$ while $\mathsf{Zero}_0(w) = false$ for $w \in \{1, 3, 4\}$.

At the beginning of Step t we have the matrix M_{t-1} and let us see what we need to do in Step t, when the operation $change_{ij}(v)$ happens: here we are given the new value v of m_{ij} and say, the previous value of m_{ij} is u. So the new matrix $M_t = M_{t-1} + Z$, where Z is the all-zeros matrix except for its (i, j)-th coordinate that is $v - u$. Let E'_{t-1} denote the matrix $U_{t-1}(M_{t-1} + Z)$.

Let the symbols $r(X, s)$ and $c(X, \ell)$ denote the s-th row and ℓ-th column of matrix X, respectively. E'_{t-1} is the same as E_{t-1}, except for its j-th column which is $c(E_{t-1}, j) + (v - u)c(U_{t-1}, i)$. Even if we assume that we had the matrix E_{t-1} with us at the beginning of Step t, we now need to "clean up" E'_{t-1} by elementary row operations to obtain E_t in our special form and this could take as much as $\Theta(n^2)$ time. (Repeating these row operations on U_{t-1} yields U_t.) Thus we will not be able to maintain the matrices U_k, E_k at the end of Step k, for each k.

Thus at the beginning of Step t we in fact do not know the matrices U_{t-1} and E_{t-1}. However we will know certain important entries of E_{t-1} and U_{t-1}. For this purpose, we need to recall the functions π and Zero defined earlier; $\pi_{t-1}(i)$ is 0 if column i is a dirty column of E_{t-1}, else it gives the row coordinate of the unique non-zero element of column

i; $\mathsf{Zero}_{t-1}(i)$ is *true* if row i is an all-zeros row in E_{t-1}, else $\mathsf{Zero}_{t-1}(i)$ is *false*. What we will ensure at the beginning of Step t (to process $change_{ij}(\cdot)$) is the following:

- we will know the entire $\pi_{t-1}(j)$-th row of E_{t-1}: this is $r(E_{t-1}, \pi_{t-1}(j))$
- we will know the i-th column of U_{t-1} in its $\pi_{t-1}(j)$-th coordinate and all those coordinates w such that $Zero_{t-1}(w) = true$; we call this the *sub-column* $\tilde{c}(U_{t-1}, i)$.

We describe our algorithm in Section 3 and show that these entries suffice to determine the rank of E'_{t-1}. Now in Step t we also compute certain rows of E_t and certain entries of the matrix U_t. We postpone the work of computing the other rows of E_t and the other entries of U_t for now and this postponed work will be executed in batches at various points in the algorithm. The technique of postponing work for a while and executing this postponed work in batches at some other time in the algorithm has been used for other problems too (for instance, for computing maximum matchings in [6]). The novelty in our paper is that when we run our update step (to compute certain entries of E_t and U_t), we do not update entire columns since that is too expensive. Instead, we are able to identify these columns in their "critical" coordinates and update columns only in their critical coordinates. We are able to identify these critical coordinates in advance due to the function π on column indices of E and the function Zero on row indices of E that we maintain throughout our algorithm and also due to the lookahead on location indices that our problem assumes.

3 The algorithm for dynamic matrix rank

Let us assume that we have a lookahead of up to s locations; for convenience let s be a power of 2. After the preprocessing step, our algorithm can process $2s$ change operations as follows: let $change_{ij}(v)$ be the t-th change operation, where $1 \le t \le 2s$. When this change operation occurs, we do the following:

- ∗ first call $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$; this returns the rank of the matrix M_t.
- ∗ if $t < 2s$ then call $update(\{t+1, \ldots, t+k\})$ where k is the largest power of 2 that divides t. This subroutine computes the rows $\pi_t(j_{t+1}), \ldots, \pi_t(j_{t+k})$ of E_t and the *sub-columns* $i_{t+1}, \ldots, i_{t+k}$ of U_t, where $(i_{t+1}, j_{t+1}), \ldots, (i_{t+k}, j_{t+k})$ are the location indices of the change operations in Steps $t+1, \ldots, t+k$.

We consider processing $2s$ change operations as described above as one *phase*. Each phase starts with the preprocessing step of computing the matrices U and E corresponding to the current M so that $UM = E$. Then we process $2s$ change operations. This finishes one phase. We will show in Section 4 that the $update(\{t+1, \ldots, t+k\})$ subroutine takes $O(nk^{\omega-1})$ time. Thus the total running time, $T(2s)$, for all the *update* subroutines in a phase is given by: $T(2s) = O(ns^{\omega-1} + 2n(s/2)^{\omega-1} + 4n(s/4)^{\omega-1} + \ldots + sn(s/s)^{\omega-1})$, which is $O(ns^{\omega-1})$.

In Section 3.1 we describe the $rank()$ subroutine and show that its running time is $O(n)$. Hence the time for processing $2s$ change operations in a phase, after the initialization step, is $O(ns^{\omega-1})$. We incur a cost of $O(n^{\omega})$ per phase for the initialization step and for every phase other than the first, let us distribute the cost of the initialization step of that phase among the $2s$ change operations of the previous phase. Thus the amortized cost of processing each change is $O(n^{\omega}/s + ns^{\omega-2})$. With $s = \Theta(n)$, our algorithm has a cost of $O(n^{\omega-1})$ per change, which proves the following theorem.

THEOREM 1. *Dynamic matrix rank over an arbitrary field can be solved using amortized $O(n^{\omega-1})$ arithmetic operations per change (where $\omega < 2.376$) with a preprocessing cost of $O(n^{\omega})$, provided that we are allowed a lookahead of up to $\Theta(n)$ locations.*

3.1 The subroutine $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$

The *rank* subroutine is called after every change operation. Let $change_{ij}(v)$ be the current change operation - this changes the matrix M_{t-1} to M_t. The input to $rank()$ consists of i, j, v, the $\pi_{t-1}(j)$-th row of E_{t-1}, and the i-th column of U_{t-1} restricted to certain critical coordinates. Let u be the value of (i, j)-th coordinate of M_{t-1}. We can assume that the new value $v \neq u$ since $M_t = M_{t-1}$ if $v = u$.

Recall that we defined E'_{t-1} to be the matrix $U_{t-1}M_t$. Below we determine the rank of E'_{t-1} and it is easy to see that $rank(M_t) = rank(E'_{t-1})$ since U_{t-1} is just a transformation matrix that is a product of elementary row operations (adding a scalar multiple of one row to another row).

Let the rank of M_{t-1} be ρ. Then the rank of M_t is one of $\rho - 1, \rho, \rho + 1$. To decide which of these 3 numbers is the rank of M_t, we need to read only those entries of E'_{t-1} as given by checks (1), (2), and (3) below. It can be shown that these entries suffice to determine the rank of E'_{t-1}.

(1) We first check if there exists any row index w such that $\mathsf{Zero}_{t-1}(w) = true$ and the w-th coordinate of $c(U_{t-1}, i)$ is non-zero.

CLAIM 2. *If j is a dirty column in E_{t-1}, then $rank(E'_{t-1}) = \rho + 1$ if there is a w such that $\mathsf{Zero}_{t-1}(w) = true$ and the w-th coordinate of $c(U_{t-1}, i)$ is non-zero; else $rank(E'_{t-1}) = \rho$.*

Thus the case when j is a dirty column in E_{t-1} (i.e., $\pi_{t-1}(j) = 0$) is easy. Just knowing those coordinates w of column $c(U_{t-1}, i)$ such that $\mathsf{Zero}_{t-1}(w) = true$ suffices to determine the rank of E_t. The case when j is a clean column is only a little more difficult. If $\pi_{t-1}(j) \neq 0$, then we also do the checks as given by (2) and (3).

(2) We check if there exists any index d such that $\pi_{t-1}(d) = 0$ and the d-th coordinate of $r(E_{t-1}, \pi_{t-1}(j))$ is non-zero.

(3) If there is neither a d of check (2) nor a w of check (1), then we check if $E'_{t-1}[\pi_{t-1}(j), j]$ is non-zero or 0. This tells us if the $\pi_{t-1}(j)$-th row of E_t will be all-zeros or not.

CLAIM 3. *If j is a clean column in E_{t-1}, then we have the following cases:*

(i) If there is a w with $\mathsf{Zero}_{t-1}(w) = true$ and $U_{t-1}[w, i] \neq 0$ <u>and</u> if there is a d with $\pi_{t-1}(d) = 0$ and $E_{t-1}[\pi_{t-1}(j), d] \neq 0$, then the rank of E'_{t-1} is $\rho + 1$.

(ii) If there is no w with $\mathsf{Zero}_{t-1}(w) = true$ and $U_{t-1}[w, i] \neq 0$ and *the row $\pi_{t-1}(j)$ in E'_{t-1} is all 0's, then the rank of E'_{t-1} is $\rho - 1$.*

(iii) Else the rank of E'_{t-1} is ρ.

Thus at the end of this step we know the rank of M_t. In summary, note that we did not really need to know the entire column $c(U_{t-1}, i)$ here - its entries in coordinates w such that $\mathsf{Zero}_{t-1}(w) = true$ and in its $\pi_{t-1}(j)$-th coordinate are what we needed; also we needed to know the row $r(E_{t-1}, \pi_{t-1}(j))$ in the dirty column coordinates and in its $\pi_{t-1}(j)$-th coordinate in order to know if this row remains a non-zero row or if it becomes the all-zeros row in

E_t. Thus the two vectors: $r(E_{t-1}, \pi_{t-1}(j))$ and $\tilde{c}(U_{t-1}, i)$ were sufficient for us to determine the rank of M_t.

Now we compute the functions π_t and Zero_t from the functions π_{t-1} and Zero_{t-1}. The only values w for which $\pi_{t-1}(w)$ and $\pi_t(w)$ might be possibly different are $w = j$, $w = d$ (where d is a dirty column in E_{t-1} but will be a clean column in E_t). The only rows r for which $\mathsf{Zero}_t(r)$ and $\mathsf{Zero}_{t-1}(r)$ might be possibly different are row $\pi_{t-1}(j)$ and row w (where w is a zero row in E_{t-1} but will be a non-zero row in E_t). We omit the details of computing the functions π_t and Zero_t here.

It is easy to see that the time taken by the *rank* subroutine is $O(n)$. Hence the following lemma can be concluded.

LEMMA 4. *The subroutine* $rank(i, j, v, r(E_{t-1}, \pi_{t-1}(j)), \tilde{c}(U_{t-1}, i))$ *computes the rank of the matrix* M_t *in* $O(n)$ *time. It also maintains the functions* π_t *(on column indices) and* Zero_t *(on row indices).*

4 The $update()$ subroutine

In each update subroutine, we will compute certain rows of E_t and certain columns of U_t in "critical coordinates" that will be useful in the next few *change* operations. In particular, we will compute k rows of E_t and k sub-columns of U_t, where $k \geq 1$ is the largest power of 2 that is a divisor of t (recall that the current change operation is the t-th change operation in this phase).

Let the change operations that will occur in in the next k steps be in the locations $(x_1, y_1), \ldots, (x_k, y_k)$, respectively. Note that we know $(x_1, y_1), \ldots, (x_k, y_k)$ due to the lookahead allowed to us. Define the set $\mathcal{S}_t = \{\pi_t(y_1), \ldots, \pi_t(y_k), o_1, \ldots, o_h\}$ where $o_1, \ldots, o_h$ are all the row indices o such that $\mathsf{Zero}_t(o) = true$. The set $\mathcal{S}_t$ is the set of critical coordinates for $update(\{t+1, \ldots, t+k\})$.

In $update(\{t+1, \ldots, t+k\})$, we will compute the k rows $\pi_t(y_1), \ldots, \pi_t(y_k)$ of E_t (note that this implies that we know all those rows s of E_t, for $s \in \mathcal{S}_t$ since rows $o_1, \ldots, o_h$ are all-zeros in E_t) and the k columns $x_1, \ldots, x_k$ of U_t in the coordinates s for $s \in \mathcal{S}_t$. Once we compute the above rows and sub-columns, we will store them so that we can use/reuse them at later steps of this phase. In our current *update* subroutine, we will be reusing the rows and sub-columns that we computed in the *update* subroutine of Step $t - k$.

Let us see what rows and sub-columns were computed in the *update* subroutine of Step $t - k$. Let us use the symbol γ to denote $t - k$. Let $change_{i_\ell j_\ell}(v_\ell)$ be the change operation in Step ℓ, for $\gamma + 1 \leq \ell \leq t$. Since the number k is the largest power of 2 that is a divisor of t, it follows that $\gamma = t - k$ is a multiple of $2k$. Hence the set of critical coordinates for Step γ, call it $\mathcal{S}_\gamma$, contains $\{\pi_\gamma(j_{\gamma+1}), \ldots, \pi_\gamma(j_t), \pi_\gamma(y_1), \ldots, \pi_\gamma(y_k), z_1, \ldots, z_g\}$ where $z_1, \ldots, z_g$ are the row indices for which Zero_γ is true. We have the following claim stated as Proposition 5. Its proof is omitted here.

PROPOSITION 5. $\mathcal{S}_t \subseteq \mathcal{S}_\gamma$.

Since the *update* subroutine of Step t computes the rows s of E_t for $s \in \mathcal{S}_t$, it follows that the *update* subroutine of Step γ computed the rows s' of E_γ for $s' \in \mathcal{S}_\gamma$. Lemma 6 follows from this fact and Proposition 5.

LEMMA 6. *The* $update$ *subroutine of Step γ computes the rows $\pi_t(y_1), \ldots, \pi_t(y_k)$ of E_γ and the columns $x_1, \ldots, x_k$ of U_γ restricted to the coordinates of $\mathcal{S}_t$.*

Now we are in a position to specify what tasks have to be performed in our current update subroutine, i.e., $update(\{t+1, \ldots, t+k\})$. Here we have to perform the following tasks:

(i) update the row $r(E_\gamma, \pi_t(y_h))$ to $r(E_t, \pi_t(y_h))$, for $1 \leq h \leq k$

(ii) update the sub-column $\tilde{c}(U_\gamma, x_h)$ to $\tilde{c}(U_t, x_h)$, for $1 \leq h \leq k$ (all these sub-columns are restricted to the coordinates $s \in \mathcal{S}_t$)

Theorem 7 is our main tool here to perform the updates given by **(i)** and **(ii)** above. During each Step ℓ, where $\gamma + 1 \leq \ell \leq t$, recall that the matrix $M_{\ell-1}$ gets changed to M_ℓ by a $change_{i_\ell j_\ell}(v_\ell)$ operation; we have $U_{\ell-1}M_\ell = E'_{\ell-1}$. In order to "clean" the matrix $E'_{\ell-1}$ we might need to clean up to 2 columns (column j_ℓ and a dirty column d_ℓ) of $E'_{\ell-1}$. The cleaning of column j_ℓ will be done by the row $\pi_\ell(j_\ell)$ and the cleaning of column d_ℓ will be done by the row $\pi_{\ell-1}(j_\ell)$. Let us use the symbols a_ℓ and b_ℓ for $\pi_{\ell-1}(j_\ell)$ and for $\pi_\ell(j_\ell)$, respectively. The row operations performed on $E'_{\ell-1}$ have to be then performed on $U_{\ell-1}$ and this yields U_ℓ.

Let us use the symbol $\tilde{E}'_{\ell-1}$ (similarly, $\tilde{U}_{\ell-1}$) to denote the matrix $E'_{\ell-1}$ (resp., $U_{\ell-1}$) *after* the "cleaning" of column j_ℓ and *before* the "cleaning" of column d_ℓ. Let e_{j_ℓ} denote the unit vector with a 1 in its j_ℓ-th coordinate.

Note that the equalities given in Theorem 7 hold for all row indices $s \in \{1, \ldots, n\}$, however we focus only on row indices $s \in \mathcal{S}_t$ here. (For simplicity of exposition, we will not qualify statements on a row $\pi_\ell(s)$ of E with "if $\pi_\ell(s) \neq 0$", thus we might refer to a row r' where $r' = 0$ - such a row will be the all 0's row.) The proof of Theorem 7 is omitted here.

THEOREM 7. *For each $s \in \mathcal{S}_t$, we have the following relation between row s of E_t and row s of E_γ:*

$$r(E_t, s) = r(E_\gamma, s) - \sum_{\ell=\gamma+1}^{t} \delta_{s,\ell} \cdot e_{j_\ell} - \sum_{\ell=\gamma+1}^{t} \alpha_{s,\ell} \cdot r(\tilde{E}'_{\ell-1}, a_\ell)$$

and the following relation between row s of U_t and row s of U_γ:

$$r(U_t, s) = r(U_\gamma, s) - \sum_{\ell=\gamma+1}^{t} \beta_{s,\ell} \cdot r(U_\ell, b_\ell) - \sum_{\ell=\gamma+1}^{t} \alpha_{s,\ell} \cdot r(\tilde{U}_{\ell-1}, a_\ell)$$

where the scalars $\delta_{s,\ell}, \alpha_{s,\ell}$ and $\beta_{s,\ell}$ are defined as follows:

$*$ *If $b_\ell = 0$, then $\delta_{s,\ell} = (u_\ell - v_\ell) \cdot U_{\ell-1}[s, i_\ell]$.*

$$Else \quad \delta_{s,\ell} = \begin{cases} (u_\ell - v_\ell) \cdot U_{\ell-1}[b_\ell, i_\ell] & \textit{if } s = b_\ell \\ E_{\ell-1}[s, j_\ell] & \textit{otherwise} \end{cases}$$

$*$ *If $b_\ell = 0$ then $\beta_{s,\ell} = 0$.*

$$Else \quad \beta_{s,\ell} = \begin{cases} 0 & \textit{if } s = b_\ell \\ E'_{\ell-1}[s, j_\ell] / E'_{\ell-1}[b_\ell, j_\ell] & \textit{otherwise} \end{cases}$$

$*$ *If $a_\ell = 0$ or $a_\ell = b_\ell$ then $\alpha_{s,\ell} = 0$.*
Else let d_ℓ be the leading element of row a_ℓ in E_ℓ, i.e., $\pi_\ell(d_\ell) = a_\ell$. We have:

$$\alpha_{s,\ell} = \begin{cases} 0 & \text{if } s = a_\ell \\ E'_{\ell-1}[s, d_\ell] / E'_{\ell-1}[a_\ell, d_\ell] & \text{otherwise} \end{cases}$$

The above theorem can be written in matrix form as follows. For simplicity let us call the elements of $\mathcal{S}_t$ as $s_1, \ldots, s_p$, and the $\delta_{s,\ell}$, $\beta_{s,\ell}$ and $\alpha_{s,\ell}$ values for $s \in \mathcal{S}_t$ and $1 \le \ell \le k$ as $\delta_{1,1}, \ldots, \delta_{p,k}$, $\beta_{1,1}, \ldots, \beta_{p,k}$ and $\alpha_{1,1}, \ldots, \alpha_{p,k}$, respectively.

$$\begin{bmatrix} r(E_t, s_1) \\ \vdots \\ r(E_t, s_p) \end{bmatrix} = \begin{bmatrix} r(E_\gamma, s_1) \\ \vdots \\ r(E_\gamma, s_p) \end{bmatrix} - \begin{bmatrix} \delta_{1,1} & \alpha_{1,1} & \ldots & \delta_{1,k} & \alpha_{1,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \delta_{p,1} & \alpha_{p,1} & \ldots & \delta_{p,1} & \alpha_{p,k} \end{bmatrix} \begin{bmatrix} e_{j_{\gamma+1}} \\ \vdots \\ r(\tilde{E}'_{t-1}, a_t) \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} r(U_t, s_1) \\ \vdots \\ r(U_t, s_p) \end{bmatrix} = \begin{bmatrix} r(U_\gamma, s_1) \\ \vdots \\ r(U_\gamma, s_p) \end{bmatrix} - \begin{bmatrix} \beta_{1,1} & \alpha_{1,1} & \ldots & \beta_{1,k} & \alpha_{1,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{p,1} & \alpha_{p,1} & \ldots & \beta_{p,k} & \alpha_{p,k} \end{bmatrix} \begin{bmatrix} r(U_\gamma, b_{\gamma+1}) \\ \vdots \\ r(\tilde{U}_{t-1}, a_t) \end{bmatrix} \quad (2)$$

Our goal is to determine the matrices on the left of Eqns. (1) and (2). However notice that these matrices can be quite large. Each matrix on the left is a $p \times n$ matrix, where $p = |\mathcal{S}_t|$. The value of $|\mathcal{S}_t|$ could be $\Theta(n)$ and then we would spend $\Theta(n^2)$ time only to just write down all the entries of such a matrix. We certainly do not want to spend $\Theta(n^2)$ time for *update*$(\{t+1, \ldots, t+k\})$. Recall that we promised to show that *update*$(\{t+1, \ldots, t+k\})$ takes $O(nk^{\omega-1})$ time. Hence we do not perform all the matrix arithmetic as specified by Eqns. (1) and (2).

Instead, to compute the relevant rows of E_t, we restrict the matrices of Eqn. (1) solely to the row indices $\pi_t(y_1), \ldots, \pi_t(y_k)$ since job (i) only needs these rows of E_t. This involves multiplying a $k \times 2k$ matrix (of α's and δ's) with a $2k \times n$ matrix which takes $O(nk^{\omega-1})$ time, once we know the matrices involved. To compute the sub-columns of U_t, we restrict the matrices of Eqn. (2) only to the column indices $x_1, \ldots, x_k$, since job (ii) only needs columns $x_1, \ldots, x_k$ of U_t, restricted to coordinates in $\mathcal{S}_t$. This involves multiplying a $p \times 2k$ matrix (of α's and β's) with a $2k \times k$ matrix of sub-rows, which again takes $O(nk^{\omega-1})$ time. Observe that we need the entire $p \times 2k$ matrix of β's and α's written above for this matrix multiplication. Determining the β's, α's, and δ's is, in fact, the crux of the *update* subroutine. We will show the following lemma here. Section 4.1 contains the algorithm that proves this lemma.

LEMMA 8. *The $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $\gamma + 1 \le \ell \le t$ can be computed in time $O(nk^{\omega-1})$.*

Once we compute the matrix of α's and δ's, task (i) of *update*$(\{t+1, \ldots, t+k\})$ is essentially done since we now know all the matrices on the right of Eqn. (1): the matrix whose rows are $r(E_\gamma, s)$ for $s \in \{\pi_t(y_1), \ldots, \pi_t(y_k)\}$ is known to us (by Lemma 6) and each *odd* indexed row in the rightmost matrix is a unit vector (e_{j_ℓ} is the $(2(\ell - \gamma) - 1)$-th row). Regarding the *even* indexed rows, the vector $r(E'_{\ell-1}, a_\ell)$ was a part of the input to the *rank* subroutine of Step ℓ (recall that $a_\ell = \pi_{\ell-1}(j_\ell)$) and we have $r(\tilde{E}'_{\ell-1}, a_\ell) = r(E'_{\ell-1}, a_\ell) - \beta_{a_\ell,\ell} \cdot e_{j_\ell}$.

Completing task (ii) of $update(\{t+1,\dots,t+k\})$ is more difficult, the rightmost matrix of Eqn. (2) is unknown to us. We describe how we compute this matrix after we present the proof of Lemma 8.

4.1 Proof of Lemma 8

We compute the $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $\gamma+1 \le \ell \le t$ using a recursive subroutine UpdateColumns($\{\gamma+1,\dots,t\}$). This subroutine computes these scalars by determining the sub-columns $\tilde{c}(U_{\ell-1}, i_\ell), \tilde{c}(E_{\ell-1}, j_\ell), \tilde{c}(E_{\ell-1}, d_\ell)$ in the coordinates of $\mathcal{S}_t$ for $\gamma+1 \le \ell \le t$; recall that these sub-columns determine the $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values by Theorem 7.

Here we describe a generic subroutine UpdateColumns($\{w+1,\dots,w+q\}$) (this is either the original subroutine UpdateColumns($\{\gamma+1,\dots,t\}$) or a subroutine invoked in a recursive call). This subroutine will compute $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for all $s \in \mathcal{S}_t$ and $w+1 \le \ell \le w+q$. We will maintain the invariant that we know (restricted to the coordinates of $\mathcal{S}_t$) the sub-columns j_ℓ, d_ℓ of E_w and the sub-columns i_ℓ of U_w, for $w+1 \le \ell \le w+q$, at the time UpdateColumns($\{w+1,\dots,w+q\}$) is called. Observe that this invariant is true at the onset when UpdateColumns($\{\gamma+1,\dots,t\}$) is called, since we know the vectors $\tilde{c}(E_\gamma, j_\ell)$, $\tilde{c}(E_\gamma, d_\ell)$, $\tilde{c}(U_\gamma, i_\ell)$ in the coordinates of $\mathcal{S}_t$ for $\gamma+1 \le \ell \le t$ (by the *update* subroutine of Step γ and because $\mathcal{S}_\gamma \supseteq \mathcal{S}_t$).

```
UpdateColumns($\{w+1,\dots,w+q\}$):
  • If $q = 1$ then it is the base case: we compute the values $\delta_{s,w+1}$, $\beta_{s,w+1}$ and $\alpha_{s,w+1}$
    for all $s \in \mathcal{S}_t$ from $\tilde{c}(E_w, j_{w+1})$, $\tilde{c}(E_w, d_{w+1})$, $\tilde{c}(U_w, i_{w+1})$, and scalars $v_{w+1}, u_{w+1}$ using
    Theorem 7.
  • Else
      (1) Call UpdateColumns($\{w+1,\dots,w+q/2\}$) recursively.
      (2) Perform a bulk update step to update
            (I) the sub-columns $j_\ell, d_\ell$ of $E_w$ to $E_{w+q/2}$ for $w+q/2+1 \le \ell \le w+q$
            (II) the sub-columns $i_\ell$ of $U_w$ to $U_{w+q/2}$ for $w+q/2+1 \le \ell \le w+q$
      (3) Call UpdateColumns($\{w+q/2+1,\dots,w+q\}$) recursively.
```

Remark. Observe that Step (2) enables us to maintain the following invariant that is necessary when UpdateColumns($\{w+q/2+1,\dots,w+q\}$) is called in Step (3): we know the columns $\tilde{c}(E_{w+q/2}, j_\ell), \tilde{c}(E_{w+q/2}, d_\ell), \tilde{c}(U_{w+q/2}, i_\ell)$ in the coordinates of $\mathcal{S}_t$ for $w+q/2+1 \le \ell \le w+q$.

The base case is easy since we had maintained the invariant that we know the sub-columns $\tilde{c}(E_w, j_{w+1})$, $\tilde{c}(E_w, d_{w+1})$, $\tilde{c}(U_w, i_{w+1})$ in the coordinates s, where $s \in \mathcal{S}_t$, when the subroutine UpdateColumns($\{w+1\}$) is called. The base case takes $O(n)$ time. Thus we have the following recurrence for the running time $T'(q)$ of UpdateColumns($\{w+1,\dots,w+q\}$):

$$T'(q) = \begin{cases} 2T'(q/2) + \text{time for the bulk update step} & \text{if } q > 1 \\ O(n) & \text{if } q = 1 \end{cases} \tag{3}$$

Now we need to describe the bulk update step. The bulk update step has to perform the updates given by (I) and (II) of Step (2) in the algorithm UpdateColumns($\{w+1,\dots,w+q\}$) described above. We describe first how we do (I) and then (II).

(I): Update columns j_ℓ, d_ℓ of E_w to $E_{w+q/2}$. Here we need to update the columns j_ℓ, d_ℓ of E_w to $E_{w+q/2}$, for $w + q/2 + 1 \le \ell \le w + q$. This amounts to updating the coordinates $j_{w+q/2+1}, \ldots, j_{w+q}, d_{w+q/2+1}, \ldots, d_{w+q}$ of certain *rows* of E_w to $E_{w+q/2}$. These row indices are the numbers s in $\mathcal{S}_t$. The updates on these rows of E_w are given by equations analogous to the ones in Theorem 7. We have for each $s \in \mathcal{S}_t$ the following equation:

$$r(E_{w+q/2}, s) = r(E_w, s) - \sum_{\ell=w+1}^{w+q/2} \delta_{s,\ell} \cdot e_{j_\ell} - \sum_{\ell=w+1}^{w+q/2} \alpha_{s,\ell} \cdot r(\tilde{E'}_{\ell-1}, a_\ell)$$

where the $\delta_{s,\ell}$'s and $\alpha_{s,\ell}$'s are defined in Theorem 7. Note that we already know all $\delta_{s,\ell}, \alpha_{s,\ell}, \beta_{s,\ell}$ values for $w + 1 \le \ell \le w + q/2$, since we computed these values in the UpdateColumns($\{w + 1, \ldots, w + q/2\}$) subroutine, that was called in Step 1 of UpdateColumns($\{w + 1, \ldots, w + q\}$). The above equations in matrix form become:

$$\begin{bmatrix} r(E_{w+q/2}, s_1) \\ \vdots \\ r(E_{w+q/2}, s_p) \end{bmatrix} = \begin{bmatrix} r(E_w, s_1) \\ \vdots \\ r(E_w, s_p) \end{bmatrix} - \begin{bmatrix} \delta_{1,1} & \ldots & \alpha_{1,q/2} \\ \vdots & \vdots & \vdots \\ \delta_{p,1} & \ldots & \alpha_{p,q/2} \end{bmatrix} \begin{bmatrix} e_{j_{w+1}} \\ \vdots \\ r(\tilde{E'}_{w+q/2-1}, a_{w+q/2}) \end{bmatrix} \tag{4}$$

where the rows of the E matrices are restricted to the coordinates $j_{w+q/2+1}, \ldots, j_{w+q}, d_{w+q/2+1}, \ldots, d_{w+q}$. Recall that $s_1, \ldots, s_p$ are the elements of $\mathcal{S}_t$ and for convenience, we called the $\delta_{s,\ell}$ and $\alpha_{s,\ell}$ values that we computed in UpdateColumns($\{w + 1, \ldots, w + q/2\}$) as $\delta_{1,1}, \ldots, \delta_{p,q/2}$ and $\alpha_{1,1}, \ldots, \alpha_{p,q/2}$.

We know all the matrices on the right in Eqn. (4) (regarding the rows of the rightmost matrix, refer to the paragraph after the statement of Lemma 8). Since we multiply a $p \times q$ matrix with a $q \times q$ matrix in Eqn. (4), the time taken for this matrix multiplication is $O((p/q)q^\omega)$, which is $O(nq^{\omega-1})$. We thus determine for each $s \in \mathcal{S}_t$, the row s of $E_{w+q/2}$ restricted to coordinates j_ℓ, d_ℓ, for $w + q/2 + 1 \le \ell \le w + q$, in $O(nq^{\omega-1})$ time; in other words, we know the columns j_ℓ, d_ℓ, for $w + q/2 + 1 \le \ell \le w + q$, of $E_{w+q/2}$, in the coordinates of elements in $\mathcal{S}_t$ in $O(nq^{\omega-1})$ time.

(II): Update columns i_ℓ of U_w to $U_{w+q/2}$. Here we need to update columns i_ℓ, for $w + q/2 + 1 \le \ell \le w + q$ of U_w to $U_{w+q/2}$. We follow the same method that we used to update the columns of E_w to $E_{w+q/2}$. For each $s \in \mathcal{S}_t$ we have the following equation:

$$r(U_{w+q/2+1}, s) = r(U_w, s) - \sum_{\ell=w+1}^{w+q/2} \beta_{s,\ell} \cdot r(U_{\ell-1}, b_\ell) - \sum_{\ell=w+1}^{w+q/2} \alpha_{s,\ell} \cdot r(\tilde{U}_{\ell-1}, a_\ell)$$

where the $\beta_{s,\ell}$'s and $\alpha_{s,\ell}$'s are defined in Theorem 7. The bulk update step for the rows of U (written in matrix form) is analogous to Eqn. (4) - note that here these rows are restricted to the coordinates i_ℓ for $w + q/2 + 1 \le \ell \le w + q$. However, here we cannot claim that we know all the matrices on the right of the analogous equation of Eqn. (4) for U. The rightmost matrix, whose rows are the sub-rows $r(U_w, b_{w+1}), r(\tilde{U}_w, a_{w+1}), \ldots, r(\tilde{U}_{w+q/2-1}, a_{w+q/2})$ is unknown to us and we have to determine it now. We will compute the rows of this matrix using the same recursive strategy as was used in the UpdateColumns algorithm. The subroutine UpdateRowsU($\{w + 1, \ldots, w + q/2\}$), described below, computes these rows.

We present a generic subroutine UpdateRowsU$(\{z+1,\ldots,z+l\})$ (this is either the original subroutine UpdateRowsU$(\{w+1,\ldots,w+q/2\})$ or one invoked in a recursive call). Note that $w+1 \le z+1 \le z+l \le w+q/2$. We maintain the invariant that at the time of calling UpdateRowsU$(\{z+1,\ldots,z+l\})$, we have the rows $b_{z+1},\ldots,a_{z+l}$ of U_z in the coordinates of $i_{w+q/2+1},\ldots,i_{w+q}$; in this subroutine we update these sub-rows to the sub-rows $r(U_z,b_{z+1}),\ldots, r(\tilde{U}_{z+l-1},a_{z+l}))$, respectively. Observe that this invariant is true at the onset when we call UpdateRowsU$(\{w+1,\ldots,w+q/2\})$ by the invariant that we had maintained when UpdateRowsU$(\{w+1,\ldots,w+q\})$ was called.

UpdateRowsU$(\{z+1,\ldots,z+l\})$:

- If $l=1$ then it is the base case: by our invariant, we have the rows $r(U_z,b_{z+1})$ and $r(U_z,a_{z+1})$ in the coordinates $i_{w+q/2+1},\ldots,i_{w+q}$. We need to update $r(U_z,a_{z+1})$ to $r(\tilde{U}_z,a_{z+1})$. This is easily done.
- Else
  - Call UpdateRowsU$(\{z+1,\ldots,z+l/2\})$.
  - Simultaneously update the sub-rows $b_{z+l/2+1},\ldots,a_{z+l}$ (call these row indices $s'_1,\ldots,s'_l$ for convenience) of U_z to $U_{z+l/2}$ as follows:

$$\begin{bmatrix} r(U_{z+l/2},s'_1) \\ \vdots \\ r(U_{z+l/2},s'_l) \end{bmatrix} = \begin{bmatrix} r(U_z,s'_1) \\ \vdots \\ r(U_z,s'_l) \end{bmatrix} - \begin{bmatrix} \beta_{1,1} & \ldots & \alpha_{1,l/2} \\ \vdots & \vdots & \vdots \\ \beta_{l,1} & \ldots & \alpha_{l,l/2} \end{bmatrix} \begin{bmatrix} r(U_z,(b_{z+1}) \\ \vdots \\ r(\tilde{U}_{z+l/2-1},a_{z+l/2}) \end{bmatrix}$$

    We know all the matrices on the right side above, since we have already computed the matrix of α's and β's corresponding to $\{z+1,\ldots,z+l\}$ during the subroutine UpdateColumns$(\{w+1,\ldots,w+q/2\})$ and the rightmost matrix was computed in the earlier recursive call UpdateRowsU$(\{z+1,\ldots,z+l/2\})$.
  - Call UpdateRowsU$(\{z+l/2+1,\ldots,z+l\})$. [Note that the update of the above step ensures that our invariant is maintained for this call of UpdateRowsU.]

It is easy to see that the recurrence relation for the running time $T''(l)$ of the above algorithm UpdateRowsU$(\{z+1,\ldots,z+l\})$ is:

$$T''(l) = \begin{cases} 2T''(l/2) + O(ql^{\omega-1}) & \text{if } l > 1 \\ O(q) & \text{if } l = 1 \end{cases}$$

$T''(l)$ solves to $O(ql^{\omega-1})$. Thus $T''(q)$ is $O(q^{\omega})$. This is the time taken to compute the rightmost matrix in the equation analogous to Eqn. (4) for U. Once this matrix is determined, the matrix multiplication is performed in $O(nq^{\omega-1})$ time. Thus we determine the sub-rows s of $U_{w+q/2}$ for $s \in \mathcal{S}_t$ in the coordinates $i_{w+q/2+1},\ldots,i_{w+q}$. In other words, we computed the columns $i_{w+q/2+1},\ldots,i_{w+q}$ of $U_{w+q/2}$ in the coordinates of $\mathcal{S}_t$. This completes the description of the bulk update step of the subroutine UpdateColumns$(\{w+1,\ldots,w+q\})$.

We can now analyse the running time $T'(q)$ of UpdateColumns$(\{w+1,\ldots,w+q\})$ (see Eqn. (3)). We have $T'(q) = 2T'(q/2) + O(nq^{\omega-1})$ and $T'(1) = O(n)$. Thus $T'(q)$ solves to $O(nq^{\omega-1})$. Hence $T'(k)$, the running time of UpdateColumns$(\{\gamma+1,\ldots,t\})$, is $O(nk^{\omega-1})$. This proves Lemma 8. ∎

Completing task (ii) of $update(\{t+1,\ldots,t+k\})$**.** We have to determine all the matrices on the right hand side of Eqn. (2) - we currently know two of these matrices: the matrix whose rows are $r(U_\gamma, s)$ restricted to columns $i_{\gamma+1},\ldots,i_t$ (by Lemma 6) and the matrix of α's and β's that we just computed. The rightmost matrix of Eqn. (2) is currently unknown to us; however we know the matrix whose rows are $b_1, a_1, \ldots, b_k, a_k$ of U_γ. We need to update this matrix to the desired matrix. This is done by calling the subroutine $\mathsf{UpdateRowsU}(\{\gamma+1,\ldots,t\})$ described in the previous section. This takes $O(nk^{\omega-1})$ time by our analysis given there. Now we know all the matrices on the right of Eqn. (2). Thus we can compute the matrix on the left of Eqn. (2) in time $O(nk^{\omega-1})$. This completes the description of the update subroutine. We have thus shown the following theorem.

THEOREM 9. *The* $update(\{t+1,\ldots,t+k\})$ *subroutine obtains the rows* $r(E_t, \pi_t(j_h))$ *and the sub-columns* $\tilde{c}(U_t, i_h)$ *in the coordinates* s *for* $s \in \mathcal{S}_t$*, for* $\gamma+1 \le h \le t$*, in time* $O(nk^{\omega-1})$*.*

Conclusions

We showed that the dynamic matrix rank problem for an $n \times n$ matrix with entries from any field can be solved using amortized $O(n^{\omega-1})$ arithmetic operations per change (where $\omega < 2.376$) with a preprocessing cost of $O(n^\omega)$, provided that we are allowed a lookahead of up to $\Theta(n)$ locations. An open problem is to show such a bound without lookahead.

References

[1] M. Berdan. A matrix rank problem. Master's thesis, University of Waterloo, December 2003.

[2] G. S. Frandsen and P. F. Frandsen. Dynamic Matrix Rank. In *Proc. of the 33rd ICALP*, LNCS 4051: 395-406, 2006.

[3] J. F. Geelen. Maximum rank matrix completion. *Linear Algebra Appl.*, 288(1-3): 211-217, 1999.

[4] N. J. A. Harvey, D. R. Karger, and K. Murota. Deterministic network coding by matrix completion. In *Proc. of the 16th Annual ACM-SODA*, 489-498, 2005.

[5] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead on dynamic graph problems. In *Proc. of the 7th Annual ACM-SIAM SODA*: 222-231, 1996.

[6] M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *Proc. of the 45th Annual IEEE FOCS*: 248-255, 2004.

[7] P. Sankowski. Dynamic Transitive Closure via Dynamic Matrix Inverse. In *Proc. of the 45th Annual IEEE FOCS*: 509-517, 2004.

[8] P. Sankowski. Faster Dynamic Matchings and Vertex Connectivity. In *Proc. of the 18th Annual ACM-SIAM SODA*: 118-126, 2007.

[9] P. Sankowski and M. Mucha. Fast Dynamic Transitive Closure with Lookahead. To appear in *Algorithmica*. Online version available at `http://www.springerlink.com/content/1q86442762411268/`