Методы обнаружения связей в метеорологических данных

Н.В. Кандыбина

Построена дискретная математическая модель дальних связей в метеорологии, в рамках модели решается задача поиска крупных регионов с сильными дальними связями. Получены алгоритмы полиномиальной сложности. Решение задачи может быть использовано в метеорологии для задач долгосрочного прогноза погоды.

Введение

Краткосрочный прогноз погоды основывается на знании параметров погоды в соседних регионах, но при попытке долгосрочного (более месяца) прогноза учесть эти связи не удается ввиду большого их числа и неточности. Открытое несколько лет назад явление Эль-Ниньо – Южное колебание определяется колебаниями системы «атмосфера — океан», суть его в том, что в районе экватора постоянное теплое течение прогревается сильнее, чем обычно, в результате пассатные ветры ослабевают и нагретая вода идет обратно. Явление Эль-Ниньо устойчиво проявляется на огромных территориях и представляет пример дальней связи, когда погодная аномалия в одном регионе приводит с задержкой к аномалии погоды в другом регионе.

Исходные данные для решения задачи представляют собой значения среднемесячной температуры в каждой из $192 \times 94 = 18048$ «клеток» Земли размером примерно 210×425 км в течение 13 лет. По известной, фактически четырехмерной, матрице $192 \times 94 \times 192 \times 94 = 18048 \times 18048$ корреляций температур клеток (одна клетка в январе, а другая в феврале) решается задача поиска январского и февраль-

ского сильнокоррелирующих объединений этих клеток. Сложность задачи заключается в переборе пары подмножеств трехмиллиардного множества. Предложен эвристический алгоритм, который находит первые N наибольших сильнокоррелирующих пар прямоугольников. Для четырехмерной задачи получены оценки сложности этого алгоритма, а для двумерного аналога этой задачи найдено неулучшаемое по сложности решение.

Полученные оценки позволяют решать на ЭВМ задачу обнаружения и прогноза по дальним связям в адекватном времени. Состоятельность модели подтверждена на реальных данных, предоставленных кафедрой метеорологии географического факультета МГУ, в качестве примера в модели найдено явление PNA (Pacific North America) влияния Эль-Ниньо на Северную Америку.

Автор выражает благодарность проф. Бабину Д.Н. за постановку задачи и ценные указания и проф. Кислову А.В. за предоставленные данные и интерпретацию полученных результатов.

Основные определения и результаты

Будем рассматривать среднемесячное значение какой-нибудь метеорологической наблюдаемой (например, температуры) в одной точке Земной поверхности как случайную величину. Тогда набор значений этой наблюдаемой за некоторый промежуток лет (например, 1976–1989) будет выборкой этой случайной величины. Для двух месяцев (например, января и февраля) и каждой пары точек на карте Земли подсчитаем коэффициент корреляции соответствующих им случайных величин и огрубим его (0 или 1) в соответствии с выбранным пороговым значением. В итоге образуется (четырехмерная) матрица корреляций, в каждой клетке которой стоит 1 или 0.

Пусть m и m' — два выбранных месяца, а p и p' — две выбранных точки карты. Тогда ξ_{pm} и $\xi_{p'm'}$ — соответствующие им случайные величины (среднемесячные температуры в данном месяце в данной точке). Расчетные формулы для коэффициента корреляции:

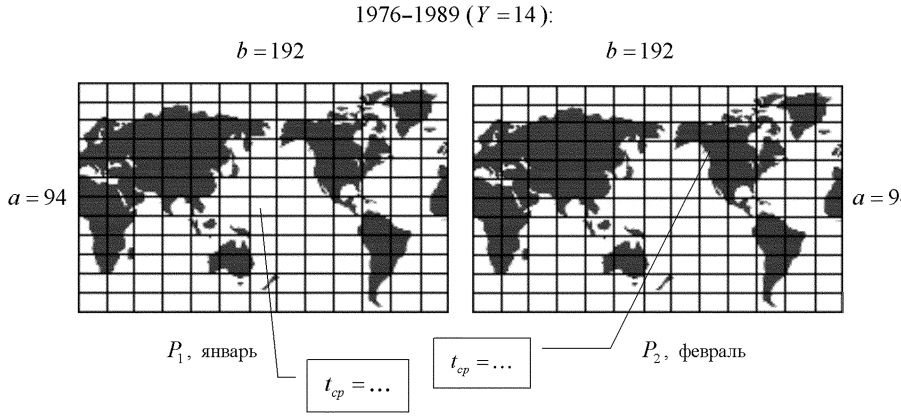

Рис. 1.

$E\xi_{pm} = \sum\limits_{y \in \mathcal{Y}} \xi^y_{pm} \cdot \frac{1}{Y}$, где $Y = |\mathcal{Y}|$ – число рассматриваемых лет

$D\xi_{pm} = (E\xi_{pm})^2 - \sum\limits_{y \in \mathcal{Y}} (\xi^y_{pm})^2 \cdot \frac{1}{Y}$

$\operatorname{cov}(\xi_{pm}, \xi_{p'm'}) = E\left(\sum\limits_{y \in \mathcal{Y}} \xi^y_{pm} \cdot \xi^y_{p'm'} \cdot \frac{1}{Y} \right) - E\xi_{pm} \cdot E\xi_{p'm'}$

$\rho(\xi_{pm}, \xi_{p'm'}) = \frac{\operatorname{cov}(\xi_{pm}, \xi_{p'm'})}{\sqrt{D\xi_{pm} \cdot D\xi_{p'm'}}}$

Выберем порог на коэффициент корреляции μ.

Дискретный коэффициент корреляции $\mu_{pp'}$ для точек p и p' определяется так: если выполняется

$$\begin{cases} \rho(\xi_{pm}, \xi_{p'm'}) & \geqslant \mu \\ \operatorname{sgn} \rho(\xi_{pm}, \xi_{p'm'}) & = \operatorname{sgn} \mu \end{cases},$$

то $\mu_{pp'} = 1$, тогда будем говорить, что между данными точками существует связь. В противном случае $\mu_{pp'} = 0$, и связи нет.

Образуется матрица $M = \{\mu_{pp'}\} = P_m \times P_{m'}$, где
$P_m = \{p = (p_1, p_2)\} = [1, a] \times [1, b]$ — карта для первого месяца (m),
$P_{m'} = \{p' = (p'_1, p'_2)\} = [1, a] \times [1, b]$ — карта для второго месяца (m'),

а $\mu_{pp'} \in \{0,1\}$ — дискретный коэффициент корреляции случайных величин, соответствующих точкам $p \in P_m$ и $p' \in P_{m'}$.

Полученная матрица корреляций будет содержать $(ab)^2 = (1.8 \cdot 10^4)^2 = 3.24 \cdot 10^8$ значений (6.5 ГБ в текстовом формате), а сложность ее подсчета составит $O(Y \cdot (ab)^2)$ по числу операций. В данной модели связи между точками p и p' соответствует единичный дискретный коэффициент корреляции $\mu_{pp'} = 1$. Будем считать, что между двумя регионами существует дальняя связь, или телеконнекция, если наблюдается попарная связь между их точками.

Пусть $M = \{\mu_{pp'}\}$, где $\mu_{pp'} = \mu(p_1, p_2, p'_1, p'_2)$, — четырехмерная матрица корреляций для месяцев m и m'. Назовем два множества из карт для этих месяцев $P \subset P_m$ и $P' \subset P_{m'}$ *коррелирующими*, если область матрицы M, являющаяся декартовым произведением P и P', содержит только единицы.

В качестве областей, отражающих явление телеконнекции, будем рассматривать прямоугольные параллелепипеды из единиц.

Область T четырехмерной матрицы $A = \{a_{ijkl}\}$, $(i,j,k,l) \in [1,I] \times [1,J] \times [1,K] \times [1,L]$, назовем *параллелепипедом*, если существуют две клетки матрицы (i_0, j_0, k_0, l_0) и $(i_0 + s_i, j_0 + s_j, k_0 + s_k, l_0 + s_l)$, где

$$s_i, s_j, s_k, s_l \in \mathbb{N}, s_i > 0, s_j > 0, s_k > 0, s_l > 0,$$

такие что

$$T = [i_0, i_0 + s_i] \times [j_0, j_0 + s_j] \times [k_0, k_0 + s_k] \times [l_0, l_0 + s_l].$$

Задача 1. Для заданной четырехмерной матрицы из нулей и единиц

$$M = \{\mu_{pp'}\} = P_m \times P_{m'} = [1,a] \times [1,b] \times [1,a] \times [1,b], \quad \forall (p,p') : \mu_{pp'} \in \{0,1\}$$

и константы $N \ll ab$ требуется найти множество $\mathfrak{T}_N$ из N первых максимальных по объему прямоугольных параллелепипедов, состоящих только из единиц:

$\mathfrak{T}_N = \{T_1, \ldots, T_N\} \subset \mathfrak{T}$, где $\mathfrak{T}$ — множество всех параллелепипедов из единиц,

$\forall T_t \in \mathfrak{T}_N : \exists (p, p') = (p_1, p_2, p'_1, p'_2)$,
$(p_1 + s_1, p_2 + s_2, p'_1 + s'_1, p'_2 + s'_2) \in P_m \times P_{m'}$,
такие что $T = [p_1, p_1 + s_1] \times [p_2, p_2 + s_2] \times [p'_1, p'_1 + s'_1] \times [p'_2, p'_2 + s'_2]$,
$\forall T_t \in \mathfrak{T}_N : \forall (p, p') \in T : \mu_{pp'} = 1$.
$\forall T_t \in \mathfrak{T} \backslash \mathfrak{T}_N : (\exists t \in [1, N], T_t \in \mathfrak{T}_N : |T| > |T_t|) \Rightarrow$
$(\exists s \in [1, N], T_s \in \mathfrak{T}_N : T \subset T_s)$.

Как двумерный аналог задачи 1 возникает задача о поиске максимальных по площади прямоугольников из единиц в $(0, 1)$-матрице:

Задача 2. Для заданной (двухмерной) матрицы из нулей и единиц

$$A = \{a_{ij}\}, (i, j) \in [1, a] \times [1, b], \forall (i, j) : a_{ij} \in \{0, 1\}$$

и константы $N \ll ab$ требуется найти множество $\mathfrak{T}_N$ из N первых максимальных по площади прямоугольников, состоящих только из единиц:

$\mathfrak{T}_N = \{T_1, \ldots, T_N\} \subset \mathfrak{T}$,
$\forall T_t \in \mathfrak{T}_N : \exists (i, j), (i + s_i, j + s_j) \in [1, a] \times [1, b]$, такие что
$T = [i, i + s_i] \times [j, j + s_j]$,
$\forall T_t \in \mathfrak{T}_N : \forall (i, j) \in T : a_{ij} = 1$.
$\forall T_t \in \mathfrak{T} \backslash \mathfrak{T}_N : (\exists t \in [1, N], T_t \in \mathfrak{T}_N : |T| > |T_t|) \Rightarrow$
$(\exists s \in [1, N], T_s \in \mathfrak{T}_N : T \subset T_s)$.

Далее приведены алгоритмы решения задач 1 и 2 с оценками сложности по числу операций и по памяти. Доказательства сформулированных ключевых теорем (вместе с описанием соответствующих алгоритмов) даны в следующем пункте.

Для задачи 2:

Алгоритм	Число операций	Требуемая память	Доказательство оценок
A_1^2	$O((ab)^3)$	$O((ab)^2)$	Теорема 1.
$(A_1^2)'$	$O((ab)^3)$	$O(N)$	Теорема 2.
A_4^2	$O(Nab)$	$O(ab)$	Теорема 3.

Для задачи 1:

Алгоритм	Число операций	Требуемая память	Доказательство оценок
A_1^4	$O((ab)^6)$	$O(N)$	Теорема 4.
A_3^4	$O(N(ab)^3)$	$O((ab)^2 \min(a, b))$	Теорема 2.

Доказательства теорем

Начнем с двухмерного случая. Прежде чем предъявить нетривиальные алгоритмы, решающие задачу 2, приведем пример простого переборного алгоритма, который обозначим A_1^2.

Каждый шаг его первой стадии устроен так: выбираем набор чисел

$$(i_0, j_0, s_i, s_j) : [i_0, i_0 + s_i] \subset [1, a], [j_0, j_0 + s_j] \subset [1, b],$$

он задает некоторый прямоугольник в матрице. Далее, осуществляем проход по множеству

$$\{(i, j) \in T\} = [i_0, i_0 + s_i] \times [j_0, j_0 + s_j]$$

всех клеток из этого прямоугольника: если элемент в данной клетке равен нулю: $a_{ij} = 0$, прекращаем проход и переходим к следующему прямоугольнику, если же он единица: $a_{ij} = 1$, то рассматриваем следующую клетку данного. Если на каком-то шаге в выбранном прямоугольнике не остается нерассмотренных клеток, это означает, что он содержит только единицы, то есть принадлежит множеству $\mathfrak{T}$. В таком случае, набор (i_0, j_0, s_i, s_j) (то есть прямоугольник T) помещается в ответ первой стадии. Таким образом, прямоугольники в ответе будут записаны набором «левый верхний угол, ширина и высота».

По окончанию работы этой части алгоритма, у нас оказывается полный список множества $\mathfrak{T}$. Вторая стадия алгоритма состоит в том, чтобы выбрать из него N самых больших по числу клеток и образовать искомое множество $\mathfrak{T}_N$. Для этого применим схему из небезызвестной сортировки «пузырьком»: возьмем последний записанный прямоугольник и сравним его по площади с соседом (понятно, как это сделать, ведь мы храним ширину и высоту каждого прямоугольника). «Победителя» этого сравнения, сравним со следующим по порядку прямоугольником и так далее. В конце концов, мы выявим один, самый большой, прямоугольник. Чтобы выявить N самых больших, нам потребуется проделать указанные операции N раз.

Для оценки сложности алгоритма A_1^2 нам понадобится знать общее число (всевозможных) прямоугольников в матрице A.

Лемма 1. *Общее число различных прямоугольников в матрице* $A = \{a_{ij}\}$, $(i,j) \in [1,a] \times [1,b]$ *равно* $\frac{ab(a+1)(b+1)}{4}$.

Доказательство. Поскольку каждый прямоугольник $T = [i_0, i_0 + s_i] \times [j_0, j_0 + s_j]$ матрицы A является декартовым произведением двух отрезков $[i_0, i_0 + s_i] \subset [1,a], [j_0, j_0 + s_j] \subset [1,b]$ на сторонах матрицы, а также любые два таких отрезка образуют некоторый прямоугольник, мы можем подсчитать количество всевозможных отрезков из $[1,a]$ и из $[1,b]$ и, перемножив эти два числа, получить требуемую оценку. Есть только один отрезок из $[1,a]$ длины a (сам $[1,a]$), отрезков длины $a-1$ уже $2([1,a-1],[2,a])$. Продолжая это рассуждение, мы получим, что для каждого значения длины $d \in [1,a]$ имеется $a-d+1$ различных отрезков, по числу точек, отстоящих от a не менее чем на $d-1$ (именно эти точки могут быть левыми концами искомых отрезков и только они). Таким образом, всего различных отрезков из $[1,a]$ будет

$$\sum_{d=1}^{a}(a-d+1) = \frac{a+1}{2} \cdot a = \frac{a(a+1)}{2},$$

по формуле для суммы арифметической прогрессии. Тогда общее число различных прямоугольников в матрице A равно

$$\frac{a(a+1)}{2} \cdot \frac{b(b+1)}{2} = \frac{ab(a+1)(b+1)}{4} \sim O((ab)^2).$$

Лемма доказана.

Теорема 1. *Сложность переборного алгоритма* A_1^2, *решающего задачу 2 для заданной (двухмерной) матрицы*

$$A = \{a_{ij}\}, (i,j) \in [1,a] \times [1,b], \forall (i,j) : a_{ij} \in \{0,1\},$$

и константы $N \ll ab$, *равна* $O((ab)^3)$ *по числу операций и* $O((ab)^2)$ *по памяти.*

Доказательство. Подсчитаем общую сложность первой стадии: каждое из чисел i_0 и s_i может быть выбрано a способами, а каждое из чисел j_0 и s_j — b способами, итого получается $(ab)^2$ вариантов.

По лемме 1 о числе прямоугольников, $\frac{ab(a+1)(b+1)}{4}$ из этих вариантов набора (i_0, j_0, s_i, s_j) будут задавать прямоугольник. Для каждого прямоугольника мы рассматриваем, в худшем случае (когда он состоит только из единиц), все его клетки, таким образом, общее число рассматриваемых клеток можно оценить как $O((ab)^3)$. Теперь разберемся с памятью: за все время выполнения первой стадии алгоритма, нам нужно одновременно хранить в памяти только текущий набор (i_0, j_0, s_i, s_j) и текущую клетку прямоугольника, который этот набор задает, получается, нам требуется только $O(1)$ мест в оперативной памяти. Для выписывания ответа, по лемме 1, нам потребуется еще порядка $O((ab)^2)$ мест в памяти.

Сложность второй стадии алгоритма, учитывая, что мощность ответа первой составляла не более $((ab)^2)$ прямоугольников, составит не более $O(N(ab)^2)$ операций. Дополнительной памяти потребуется $O(1)$, но можно использовать те же переменные, что и на первой стадии.

С учетом $N \ll ab$, общее число операций в указанном алгоритме решения задачи 2 составит $O((ab)^3)$, при этом потребуется $O((ab)^2)$ мест в памяти. Теорема доказана.

Теорема 2. *Можно сократить объем памяти, требуемой для работы алгоритма A_1^2, с $O((ab)^2)$ до $O(N)$.*

Доказательство. Совместим обе стадии алгоритма в одну. На каждом n-ом шаге первой стадии будем держать в памяти только «локальный ответ» — множество $\mathfrak{T}_N^n$ из N максимальных на данный момент прямоугольников: $\mathfrak{T}_N^n = \{T_1^n, \ldots, T_{\min_n}^n, \ldots, T_N^n\}$ и, дополнительно, номер $\min_n$ самого маленького из них. С самого начала это множество пусто: $\mathfrak{T}_N^0 = \varnothing$, и мы заполняем его первыми N состоящими только из единиц прямоугольниками, полученными в процессе выполнения алгоритма. При выявлении нового заполненного единицами прямоугольника T, определяемого набором (i_0, j_0, s_i, s_j), мы будем сравнивать его с самым маленьким из имеющихся. При «положительном» исходе сравнения: $|T| > |T_{\min_n}^n|$, то есть если $s_i \cdot s_j > s_i^{\min_n} \cdot s_j^{\min_n}$, где набор $(i_0^{\min_n}, j_0^{\min_n}, s_i^{\min_n}, s_j^{\min_n})$ как раз и задает прямоугольник $T_{\min_n}^n$, будем записывать новый прямоугольник в ответ на ме-

сто старого: $T^{n+1}_{\min_n} = T$, таким образом $\mathfrak{T}^{n+1}_N = \{T^{n+1}_1, \ldots, T^{n+1}_{\min_n} = T, \ldots, T^{n+1}_N\}$. Затем нам нужно снова отыскать самый маленький прямоугольник из $\mathfrak{T}^{n+1}_N$, что можно делать методом сравнений соседей из сортировки «пузырьком», только теперь «всплывать» будет самый маленький прямоугольник. Общая сложность данной процедуры для одного прямоугольника T составит $O(N)$. По окончанию работы алгоритма множество $\mathfrak{T}_N$ из N прямоугольников и будет являться решением задачи 2. Таким образом, общее число операций в нашем алгоритме составит не более $O((ab)^3) + O(N(ab)^2) \sim O((ab)^3)$, то есть сохранится с точностью до порядка, а вот требуемая память уменьшится до $O(N)$. Теорема доказана. Получившийся в результате алгоритм обозначим $(A^2_1)'$.

Замечание. Процедуру, описанную в доказательстве теоремы 2, можно применять в любом алгоритме решения задачи 2 (а также четырехмерной задачи 1), где подходящие прямоугольники отыскиваются последовательно. Для удобства дальнейшего использования, назовем эту операцию **соотнесением с локальным ответом**. Применение ее к каждому прямоугольнику обойдется нам, как было замечено, в $O(N)$ операций.

Существует еще некоторое множество алгоритмов, решающих задачу 2. Мы не будем приводить их все, ограничившись описанием самого удачного — A^2_4.

Идея этого алгоритма состоит в проведении некоторой предварительной обработки исходной матрицы — составлении ее шаблона, с помощью которого затем удастся существенно сократить перебор. Осуществляем проход по матрице A в таком порядке: сначала первый столбец, сверху вниз, затем второй в том же направлении и т.д., параллельно составляя ее шаблон $B = \{b_{ij}\}, (i,j) \in [1,a] \times [1,b]$. При рассмотрении каждого элемента a_{ij} матрицы записываем на место (i,j) шаблона число b_{ij} по следующим правилам:

1) если $a_{ij} = 0$, то $b_{ij} = 0$,
2) при $i = 1$: если $a_{ij} = 1$, то $b_{ij} = 1$,
3) при $i > 1$: если $a_{ij} = 1$, то $b_{ij} = b_{i-1j} + 1$.

По окончанию этой процедуры, мы получим заполненный шаблон $B = \{b_{ij}\}$, каждый элемент которого b_{ij} будет равен нулю, если

элемент исходной матрицы a_{ij} был нулем, или числу единичных элементов над a_{ij}, включая его самого, в случае $a_{ij} = 1$. Сложность этой процедуры, очевидно, $O(ab)$ операций.

Идея второй стадии алгоритма такая: для каждой клетки (i, j) каждой строки i будем искать прямоугольник высотой b_{ij} максимальной ширины, причем определим их все за два прохода по строке с использованием дополнительной памяти для b переменных. Опишем эту процедуру для строки с номером i. Напомним, что каждая клетка (i, j) шаблона «тройная», в первой ее части стоит элемент b_{ij}, одну из двух оставшихся частей условно назовем правой, а число, которое будет там стоять обозначим $c_{ij}^{\text{пр}}$, а другую — левой, там будет стоять число $c_{ij}^{\text{лев}}$. Организуем стек, он будет содержать элементы данной строки шаблона с номерами столбцов, в которых они стоят, мы будем обозначать элементы стека так: $(b_{ij_1}, j_1^n), \ldots, (b_{ij_{K_n}}, j_{K_n}^n)$, где n — номер шага, $b_{ij_k}, k \in [1, K_n]$ — некоторый элемент строки i шаблона, j_k^n — номер столбца, в котором стоит этот элемент, а $K_n \leqslant b$ — текущий номер верхнего элемента стека. Сначала проходим по строке слева направо: то есть просматриваем клетки в порядке $(i, 1), (i, 2), \ldots, (i, b)$. Для каждого следующего элемента b_{ij}:

1) если стек не пуст, сравниваем текущий элемент b_{ij} с верхним элементом в стеке $b_{ij_{K_n}}$, в противном случае, выполняем условие пункта 2,
2) если $b_{ij_{K_n}} \geqslant b_{ij}$ или стек пуст, кладем b_{ij} в стек: $b_{ij_{K_n}} = b_{ij}$, такие сравнения будем называть «неудачными»,
3) если же $b_{ij_{K_n}} < b_{ij}$, то выкидываем из стека пару $(b_{ij_{K_n}}, j_{K_n})$, записываем в правую часть клетки (i, j_{K_n}) шаблона число j : $c_{ij_{K_n}}^{\text{пр}} = j$, и, не меняя текущий элемент b_{ij}, возвращаемся к первому пункту. Такие сравнения будем называть «удачными».

После перебора всех элементов строки, выкидываем из стека все пары, которые там есть, и производим запись числа b в правые части соответствующих им клеток шаблона: $c_{ij_1}^{\text{пр}} = b, \ldots, c_{ij_{K_n}}^{\text{пр}} = b$. В результате всей этой операции мы для каждой клетки шаблона определим координату правого конца искомого прямоугольника из единиц в исходной матрице (напоминаем, это прямоугольник, который содержит

клетку (i, j), высотой с максимальный столбик из единиц над элементом a_{ij}, включая его самого, максимально возможной ширины). Поскольку «неудачных» и «удачных» сравнений в сумме не может быть больше $2b$, сложность такого прохода будет порядка $O(b)$.

Теперь осуществим проход по той же строке i шаблона в обратную сторону, справа налево, то есть просматривая клетки в порядке $(i, b), (i, b-1), \ldots, (i, 1)$, и по аналогичной схеме найдем для каждой клетки левый конец искомого прямоугольника — $c_{ij}^{\text{лев}}$. В итоге у нас в каждой клетке шаблона будут записаны три числа — высота прямоугольника, его левый нижний и правый нижний углы. Теперь пройдем по строке шаблона еще один раз, слева направо, и при проходе клетки (i, j) будем производить операцию соотнесения с локальным ответом для клетки $(i, j-1)$, если выполняется одно из условий совокупности:

$$\left[\begin{array}{l} b_{ij-1} > b_{ij} \\ c_{ij-1}^{\text{пр}} - c_{ij-1}^{\text{лев}} > c_{ij}^{\text{пр}} - c_{ij}^{\text{лев}} \end{array} \right. ,$$

а также условие

$$c_{i-1j}^{\text{пр}} - c_{i-1j}^{\text{лев}} > c_{ij}^{\text{пр}} - c_{ij}^{\text{лев}}. \qquad (*)$$

Это нужно для того, чтобы в ответ не попадали вложенные прямоугольники. В конце рассмотрения всей строки нужно будет произвести процедуру соотнесения с локальным ответом для прямоугольника, построенного на последней клетке (i, b), если для данной клетки выполняется условие $(*)$, не принимаем в расчет совокупность. При рассмотрении же последней строки $i = a$, напротив, не нужно принимать в расчет это условие, а ориентироваться только на выполнение совокупности. Для последней клетки (a, b) операцию соотнесения с локальным ответом нужно провести в любом случае. В итоге мы наберем множество $\mathfrak{T}_N$, где каждый прямоугольник из единиц будет записан набором «высота, левый нижний угол, правый нижний угол», и среди этих прямоугольников не будет ни одного пары, один из которых был бы вложен в другой.

Теорема 3. *Сложность алгоритма A_4^2, решающего задачу 2 для заданной матрицы $A = \{a_{ij}\}, (i, j) \in [1, a] \times [1, b], \forall (i, j) : a_{i,j} \in \{0, 1\}$ и константы $N \ll ab$, составляет $O(Nab)$ по числу операций и $O(ab)$ по памяти.*

Доказательство. Оценим общую сложность алгоритма A_4^2. Для нахождения элементов b_{ij} шаблона, как мы уже замечали, требуется $O(ab)$ операций, для нахождения $c_{ij}^{\text{пр}}$ и $c_{ij}^{\text{лев}}$ — еще $O(ab)$. Выполнять процедуру соотнесения с локальным ответом для найденных прямоугольников придется не чаще, чем один раз для каждой клетки в матрице (у нас будет не больше, чем по одному прямоугольнику на каждую клетку), то есть для проведения этой стадии потребуется еще не более $O(Nab)$ операций. Таким образом, алгоритм отработает за время порядка, при этом потребуется дополнительной памяти порядка $O(ab)$. Теорема доказана.

Следствие 1. *Алгоритм A_4^2 является оптимальным для решения задачи 2.*

Доказательство. Сложность алгоритма A_4^2 по числу операций имеет тот же порядок — $O(ab)$ (при фиксированной константе N), что и число элементов матрицы. Придумать алгоритм, находящий в произвольной $(0,1)$-матрице максимальный прямоугольник из единиц и работающий на порядок быстрее, нам не удастся, потому что он в любом случае должен хотя бы просмотреть все элементы заданной матрицы.

Теперь перейдем к рассмотрению четырехмерного случая, то есть непосредственно к решению задачи 1. Сначала посмотрим, насколько долго справляется с задачей переборный алгоритм, A_1^4, обобщение на четырехмерный случай алгоритма $(A_1^2)'$.

Алгоритм A_1^4 абсолютно аналогичен алгоритму $(A_1^2)'$, только теперь нам предстоит на каждом шаге выбирать набор из восьми чисел:

$$(i_0, j_0, k_0, l_0, s_i, s_j, s_k, s_l) : [i_0, i_0 + s_i] \subset [1, a], [j_0, j_0 + s_j] \subset [1, b],$$
$$[k_0, k_0 + s_k] \subset [1, a], [l_0, l_0 + s_l] \subset [1, b].$$

Все дальнейшие процедуры — проход по клеткам параллелепипеда, заданного этим набором, соотнесение его с локальным ответом, если он весь состоит из единиц — проводятся абсолютно аналогично двухмерному случаю.

Для оценки сложность алгоритма A_1^4 нам потребуется лемма о числе всевозможных прямоугольных параллелепипедов, четырехмерный аналог леммы 1.

Лемма 2. *Общее число различных прямоугольных параллелепипедов в матрице* $M = \{a_{ijkl}\}, (i,j,k,l) \in [1,a] \times [1,b] \times [1,a] \times [1,b]$, *равно* $\frac{a^2b^2(a+1)^2(b+1)^2}{4}$.

Доказательство. Доказывается эта лемма аналогично лемме 1 — редукцией к отрезкам, декартово произведение которых и представляет собой каждый прямоугольный параллелепипед четырехмерной матрицы.

Теорема 4. *Алгоритм* A_1^4, *решающий задачу 1 для заданной четырехмерной матрицы*

$$M = \{a_{ijkl}\}, (i,j,k,l) \in [1,a] \times [1,b] \times [1,a] \times [1,b],$$
$$\forall (i,j,k,l) : a_{ijkl} \in \{0,1\},$$

и константы $N \ll ab$ *имеет сложность* $O((ab)^6)$ *по числу операций и* $O(N)$ *по памяти.*

Доказательство. По лемме 2, мы построим за все время работы алгоритма $O((ab)^4)$ параллелепипедов. Для каждого из них мы рассмотрим, в худшем случае (когда он состоит только из единиц), все его клетки, таким образом, общее число рассмотренных клеток можно оценить как $O((ab)^6)$. Сложность алгоритма по памяти не изменится по сравнению с двухмерным случаем и составит $O(N)$. Теорема доказана.

К сожалению, обобщить алгоритм A_4^2 на четырехмерный случай не удастся, поскольку использующийся в нем «фокус со стеком» уже не пройдет для размерности, большей двух. Поэтому придется применить другую схему, свести четырехмерную матрицу к двумерной, только тогда появится возможность использовать указанный алгоритм.

Опишем эту процедуру для матрицы корреляций. Вспомним, что $M = P_m \times P_{m'}$, и представим каждый из этих прямоугольников отрезком. Для $P_m = \{p = (p_1, p_2)\}$: сначала выпишем точки первого столбца $(p_1, 1), p_1 = 1, 2, \ldots, a$, затем точки второго столбца $(p_1, 2), p_1 = 1, 2, \ldots, a$ и т.д., наконец, точки последнего столбца

$(p_1, b), p_1 = 1, 2, \ldots, a$. В полученном отрезке точки, которые ранее были соседями по вертикали, останутся соседними, а вот соседние по горизонтали точки будут теперь расположены на расстоянии a одна от другой. Для $P_{m'} = \{p' = (p'_1, p'_2)\}$ проделаем похожую операцию, но клетки будем выписывать по строкам - $(1, p'_2), p'_2 = 1, 2, \ldots, b$, $(2, p'_2), p'_2 = 1, 2, \ldots, b$ и т.д., наконец, $(a, p'_2), p'_2 = 1, 2, \ldots, b$. Таким образом, четырехмерная матрица корреляций M как декартово произведение $P_m \times P_{m'}$ преобразуется в обычную, двухмерную, матрицу размером $ab \times ab$. Чтобы не забывать о ее изначальной структуре поделим ее b горизонтальными и a вертикальными полосами на ab частей. Каждая такая часть $M_{p_1 p'_2}$ для фиксированных $p_2 \in [1, b], p'_1 \in [1, a]$ будет представлять собой прямоугольник размером $a \times b$:

$$M_{p_1 p'_2} = \{(p, p') = (p_1, p_2, p'_1, p'_2) | p_1 \in [1, a], p'_2 \in [1, b]\}.$$

Теперь мы можем искать прямоугольники, состоящие только из единиц, в такой, двухмерной, матрице, но они будут соответствовать уже только полоскам в изначальных плоскостях карт P_m и $P_{m'}$, поэтому нам надо придумать какой-то способ склеивать такие полоски до прямоугольников. Далее приводится алгоритм A_3^4, решающий задачу 1, который основан на этой идее.

Пусть надо решить задачу 1 для четырехмерной матрицы

$$M = \{a_{ijkl}\}, (i, j, k, l) \in [1, a] \times [1, b] \times [1, a] \times [1, b], \quad \forall (i, j, k, l) : a_{ijkl} \in \{0, 1\}$$

и константы $N \ll ab$. Применим к заданной матрице преобразование к двухмерному виду, описанное выше. Далее, проведем отдельно для каждой ее части

$$M_{jk} = \{(i, j, k, l) | i \in [1, a], l \in [1, b]\}, j \in [1, b], k \in [1, a],$$

являющейся обычной матрицей размера $a \times b$, процедуру заполнения шаблона.

Пусть выполняется неравенство $a < b$. Шаблон B_{jk}, построенный по каждой части матрицы M_{jk}, будет представлять собой трехмерный куб и будет устроен так: в каждом его элементе $(i, l, h), i \in$

$[1,a], l \in [1,b], h \in [0,a]$, будет храниться ширина максимального прямоугольника из единиц в M_{jk} высотой h, имеющего своим правым нижним углом клетку (i, l). Заполнение шаблона будет происходить в два этапа.

Осуществляем проход по матрице M_{jk} в таком порядке (сначала первый столбец, сверху вниз, затем второй и т.д.):

$$(1,1),(2,1),\ldots,(a,1),(1,2),(2,2),\ldots,(a,2),\ldots,(1,b),(2,b),\ldots,(a,b),$$

параллельно составляя ее шаблон $B_{jk} = \{b^{jk}_{ilh}\}, (i,l,h) \in [1,a] \times [1,b] \times [0,a]$, заполняя пока только клетки с $h = 0$. При рассмотрении каждого элемента a^{jk}_{il} матрицы M_{jk} записываем на место $(i,l,0)$ шаблона число b^{jk}_{il0} по следующим правилам:

1) если $a^{jk}_{il} = 0$, то, $b^{jk}_{il0} = 0$,

2) при $i = 1$: если $a^{jk}_{il} = 1$, то $b^{jk}_{il0} = 1$,

3) при $i > 1$: если $a^{jk}_{il} = 1$, то $b^{jk}_{il0} = b^{jk}_{i-1l0} + 1$.

По окончанию этой процедуры, мы получим заполненный «слой» шаблона $B_{jk} = \{b^{jk}_{ilh}\}$, где $h = 0$, каждый элемент которого b^{jk}_{il0} будет равен нулю, если элемент исходной матрицы a^{jk}_{il} был нулем, или числу единичных элементов над a^{jk}_{il}, включая его самого, в случае $a^{jk}_{il} = 1$. Сложность этой процедуры, очевидно, $O(ab)$ операций.

Фиксируем $z \in [1,a]$. На каждом шаге будем работать с, так сказать, z-срезками элементов, то есть для нас будет принципиально лишь то, больше элемент, чем z или меньше. Будем рассматривать строки получившегося «слоя $h = 0$» шаблона сверху вниз: $i = 1,2,\ldots,a$. Организуем стек, он будет содержать элементы данной строки шаблона с номерами столбцов, в которых они стоят, мы будем обозначать элементы стека так: $(b^{jk}_{il_10}, l^n_1),\ldots,(b^{jk}_{il_{S_n}0}, l^n_{S_n})$, где n — номер шага, $b^{jk}_{il_s0}, s \in [1,S_n]$ — некоторый элемент строки i шаблона, l^n_s — номер столбца, в котором стоит этот элемент, а $S_n \leqslant b$ — текущий номер верхнего элемента стека. Сначала проходим по строке слева направо: то есть просматриваем клетки в порядке $(i,1),(i,2),\ldots,(i,b)$. Для каждого следующего элемента b^{jk}_{il0}:

1) если $b^{jk}_{il0} < z$, то выкидываем из стека все находящиеся там пары $(b^{jk}_{il_10}, l^n_1), \ldots, (b^{jk}_{il_{S_n}0}, l^n_{S_n})$ и производим записи в шаблон: $b^{jk}_{il_1z} = l, \ldots, b^{jk}_{il_{S_n}z} = l$,
2) если $b^{jk}_{il0} \geqslant z$, кладем его в стек.

После перебора всех элементов строки «слоя», выкидываем из стека все пары, которые там есть и для них производим запись числа b в шаблон: $b^{jk}_{il_1z} = b, \ldots, b^{jk}_{il_{S_n}z} = b$. Ясно, что число операций по заполнению «слоя $h = z$» для одного значения $z \in [1, a]$ будет порядка $O(ab)$ (каждый элемент мы обрабатываем не более двух раз — когда кладем в стек и когда вынимаем), а общая сложность заполнения шаблона (без «слоя $h = 0$») — $O(a^2b)$.

Когда все шаблоны B_{jk} заполнены, переходим ко второй стадии алгоритма. На каждом шаге этой стадии мы должны выбрать «прямоугольную рамку»:

$$(i_0, l_0, s_i, s_l) : [i_0, i_0 - s_i] \subset [1, a], [l_0, l_0 - s_l] \subset [1, b],$$

в каждом из шаблонов B_{jk} она будет задавать некоторый прямоугольник T набором «правый нижний угол, ширина, высота». Образуем в памяти дополнительную матрицу размером $b \times a$ — $A = \{y_{jk}\}$ и будем заполнять ее нулями и единицами по такому правилу: если прямоугольник, заданный «рамкой» в шаблоне B_{jk}, содержит только единицы, то $y_{jk} = 1$, иначе — $y_{jk} = 0$. Для каждого шаблона B_{jk} эту проверку в случае $a < b$ будем осуществлять так: если $b^{jk}_{i_0l_0s_l} \geqslant s_i$, то такой прямоугольник из единиц в шаблоне есть, если же $b^{jk}_{i_0l_0s_l} < s_i$, то его нет. Для случая $a > b$ критерием существования прямоугольника из единиц будет являться выполнение неравенства $b^{jk}_{i_0l_0s_i} \geqslant s_l$ (в этом случае шаблон B_{jk} заполняется по-другому, высота и ширина меняются местами). Теперь для этой матрицы проведем поиск множества максимальных прямоугольников (то есть решим задачу 2) с помощью оптимального алгоритма A^2_4.

В качестве ответа после решения задачи 2 для матрицы A мы получим N самых больших, попарно не вложенных друг в друга прямоугольников из единиц в матрице A, записанных наборами «высота, левый нижний угол, правый нижний угол»:

$$(j_{\text{лев}} = j_{\text{пр}} = j_0, k_{\text{лев}}, k_{\text{пр}}, h) : [j_0, j_0 + h] \subset [1, b], [k_{\text{лев}}, k_{\text{пр}}] \subset [1, a].$$

Тогда найденные на данном шаге четырехмерные параллелепипеды будут задаваться наборами:

$$(i_0 = i,\ j_0 = j_0,\ k_0 = k_{\text{лев}},\ l_0 = l,\ s_i = s_i,\ s_j = h,\ s_k = k_{\text{пр}} - k_{\text{лев}},\ s_l = s_l) :$$
$$[i_0, i_0 + s_i] \subset [1, a],\ [j_0, j_0 + s_j] \subset [1, b],$$
$$[k_0, k_0 + s_k] \subset [1, a],\ [l_0, l_0 + s_l] \subset [1, a].$$

Это следует из редукции к изначальной четырехмерной структуре матрицы M. Для каждого из полученных прямоугольников (последовательно) проведем процедуру соотнесения с локальным ответом, суммарная сложность которой (для всех них) составит $O(N^2)$ операций.

Перебрав все различные «прямоугольные рамки» (а их число будет равняться числу всевозможных прямоугольников в матрице размером ab, то есть $O((ab)^2)$), мы составим окончательный ответ для задачи 1.

Теорема 5. *Алгоритм A_3^4, решающий задачу 1 для заданной матрицы*

$$M = \{a_{ijkl}\}, (i, j, k, l) \in [1, a] \times [1, b] \times [1, a] \times [1, b],$$
$$\forall (i, j, k, l) : a_{ijkl} \in \{0, 1\}$$

и константы $N \ll ab$, имеет сложность по числу операций порядка $O(N(ab)^3)$ и порядка $O((ab)^2 \min(a, b))$ по требуемой памяти.

Доказательство. Подсчитаем общую сложность первой стадии алгоритма. Чтобы хранить все шаблоны для различных частей M_{jk} большой матрицы нам потребуется $ab \cdot (ab(a + 1)) \sim O((ab)^2 a)$ мест в памяти. Поскольку мы исходили из посылки $a < b$, а в противном случае на этапе конструирования шаблона можно поменять местами ширину на высоту, всего памяти нам потребуется $O((ab)^2 \min(a, b))$. Для построения одного шаблона B_{jk} мы затрачиваем $O(ab) + O(a^2 b) \sim$

$O(a^2b)$, а всего их ab, значит, общее число операций для данной стадии алгоритма составит $O((ab)^2a)$ (в рассмотренном случае $a < b$), в общем случае $O((ab)^2 \min(a, b))$.

Подсчитаем сложность второй стадии. Число различных «прямоугольных рамок» равняется числу всевозможных прямоугольников в матрице размером ab, то есть $O((ab)^2)$. Поскольку во время этой стадии нам уже не надо ничего подсчитывать, а только сравнить два числа, вся матрица A, очевидно, будет заполнена за ab операций. Места в памяти она потребует порядка $O(ab)$. Решение задачи 2 с помощью алгоритма A_4^2 потребует, как нам уже известно, $O(Nab)$ операций и $O(ab)$ мест в памяти. Суммарная сложность проведения процедуры соотнесения с локальным для полученных прямоугольников ответом составит $O(N^2)$ операций.

Таким образом, имеем: сложность первой стадии по числу операций — $O((ab)^2 \min(a, b))$, сложность второй — $O((ab)^2) \cdot (ab + O(Nab) + O(N^2)) \sim O(N(ab)^3)$, то есть суммарно — $O(N(ab)^3)$. Что касается памяти, то больше всего, конечно, ее уйдет на хранение шаблонов на первой стадии, то есть требуемое число мест в памяти будет оцениваться как $O((ab)^2 \min(a, b))$. Теорема доказана.

Проверка модели на реальных данных

Для проверки модели на реальных данных (и подтверждения ее состоятельности) была разработана автоматизированная система для человеко-машинного поиска дальних связей, с помощью которой удалось найти некоторые известные в метеорологии влияния, в частности, так называемое явление PNA (Pacific North America), первопричиной которого является вышеупомянутое колебание Эль-Ниньо.

В качестве метеорологической наблюдаемой была выбрана температура (среднемесячные ее значения). Интервал по годам был взят с 1976 по 1989, поскольку именно на эти годы приходятся пики индекса Северо-Тихоокеанского колебания. Мы будем работать с двумя зимними месяцами — январем и февралем, при этом выявим связи «с задержкой» в месяц. Вот результаты исследований метеорологов [3]:

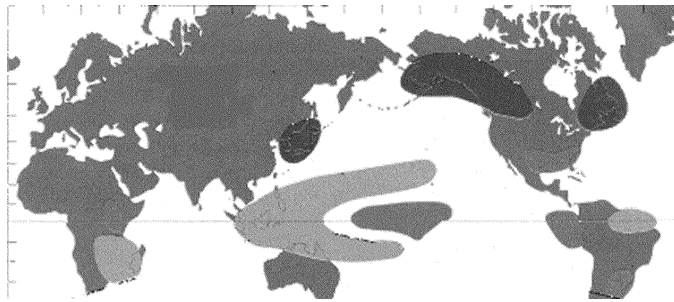

Рис. 2. Влияние явления Эль-Ниньо – Южное колебание на изменчивость метеорологического режима различных регионов Земли.

Далее (рис. 3–5) приведены полученные программой автора картинки, отражающие явление PNA. На каждом рисунке представлены две связанные области, более темная влияет на более светлую с задержкой в месяц.

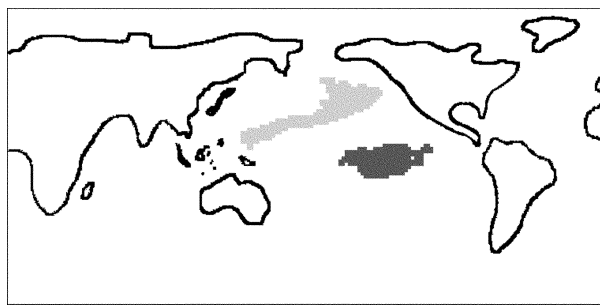

Рис. 3.

Можно попытаться уменьшить размерность данных в задаче путем объединения «похожих» (в некотором смысле) клеток в одну большую (некий регион). Для этого полезно взглянуть на распределение коэффициентов корреляций с точки зрения гладкости. Каждый выделенный регион будет определяться внутренними точками и точками границы. Для всех внутренних точек распределения коэффициента корреляции должны быть схожи, а значит, совсем немного меняться при небольшом смещении (например, на одну точку). Гра-

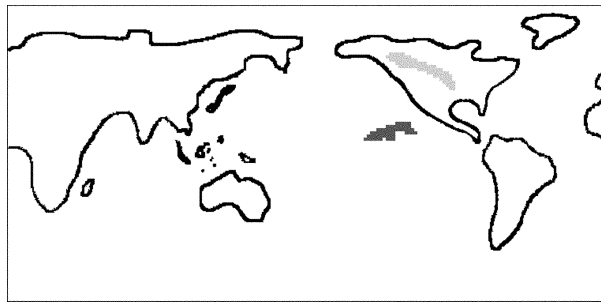

Рис. 4.

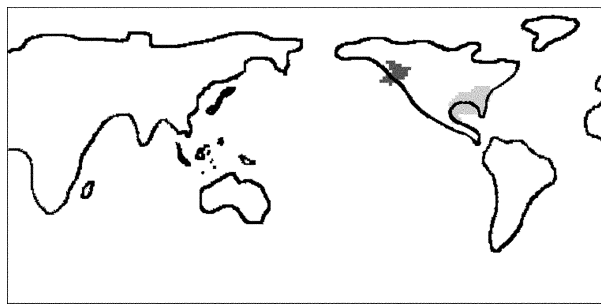

Рис. 5.

ничные точки, напротив, должны быть точками бифуркации, резкого изменения корреляционной картины. Строить эту картину областей и границ будем так: сначала зафиксируем два месяца (возможно, совпадающих) m и m' и высчитаем полную матрицу корреляций для этой пары месяцев. Затем для каждой точки $p = (p_1, p_2) \in P_m$ карты определим множество $S(p)$ соседних[1] точек по вертикали и горизонтали:

$$S(p) = S(p_1, p_2) = \{(p_1 - 1, p_2), (p_1 + 1, p_2), (p_1, p_2 - 1), (p_1, p_2 + 1), \}$$

и вычислим функцию:

[1]Соседство определяется с учетом того, что наша карта представляет собой развертку цилиндра.

$$f(p) = f(p_1, p_2) = \frac{1}{ab|S(p)|} \sum_{\bar{p} \in S(p)} \sum_{p' \in P_{m'}} |\mu_{pp'} - \mu_{\bar{p}p'}|,$$

где ab — общее число точек на карте ($a = 192, b = 94$). Тогда чем меньше функция $f(p)$, тем менее отличается распределение коэффициента корреляции для данной точки от распределения для ее соседей.

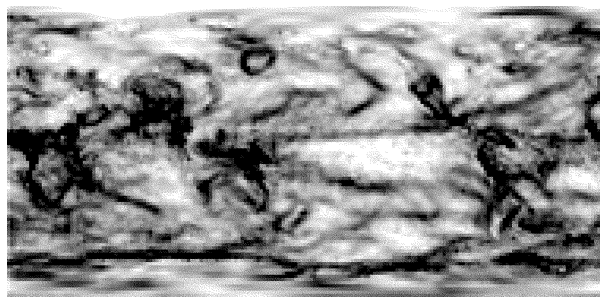

Рис. 6.

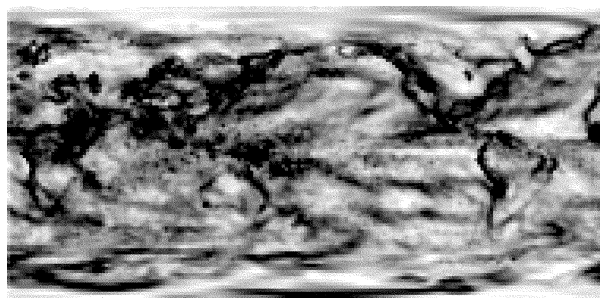

Рис. 7.

На рисунках 6 и 7 представлены такие изображения для корреляций январь–февраль и апрель–май соответственно, где первый месяц означает, откуда брались точки, а второй — на какой месяц строилась для них корреляционная картина. Общее правило: чем светлее точка, тем меньше $f(p)$, и тем более похожа ее корреляционная картина на картину для соседних точек. Напротив, черные точки — это точки с большим значением $f(p)$, «неустойчивые».

Во-первых, заметим, что горные массивы (Гималаи, Анды, центральная часть Африки), а также большая часть контуров материков — черные. Это логично, поскольку температурный режим высокогорных районов отличается от равнинного, а температурный режим суши — от океанского. Во-вторых, на первой картинке мы видим, что область Эль-Ниньо выделяется большим светлым пятном, это соответствует, видимо, как раз вышеупомянутому языку нагретой океанской воды. В-третьих, на этой же картинке более или менее хорошо видно белое пятно на западном побережье Северной Америки, вероятно, отражение влияния PNA. Что характерно, для картинки апрель–май (второй рисунок), большое белое пятно на месте Эль-Ниньо исчезает, да и вся картина гладкости претерпевает существенные изменения, что логично, поскольку в весенние месяцы погоду определяют другие закономерности.

Список литературы

[1] Носов В.А. Основы теории алгоритмов и анализа их сложности. М., 1992.

[2] Кислов А.В. Теория климата. М., 1989.

[3] Trenberth K.E., Branstator G.W., Karoly D., Kumar A, Lau N.-C., Ropelewski C. Progress during TOGA in understanding and modeling global teleconnections associated with tropical sea surface temperatures. Journal of Geophysical Research. 1998.