Theoretical Computer Science 525 (2014) 138–149

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Efficient all path score computations on grid graphs

Ury Matarazzo [1], Dekel Tsur [1], Michal Ziv-Ukelson *,[2]

Department of Computer Science, Ben-Gurion University of the Negev, Israel

ARTICLE INFO

Keywords:
Sequence alignment
All path score computations

ABSTRACT

We study the *Integer-weighted Grid All Paths Scores* (IGAPS) problem, which is given a grid graph, to compute the maximum weights of paths between every pair of a vertex on the first row of the graph and a vertex on the last row of the graph. We also consider a variant of this problem, periodic IGAPS, where the input grid graph is periodic and infinite. For these problems, we consider both the general (dense) and the sparse cases.

For the sparse periodic IGAPS problem with 0–1 weights, we give an $O(r \log^3(n^2/r))$ time algorithm, where r is the number of (diagonal) edges of weight 1. Our result improves upon the previous $O(n\sqrt{r})$ result by Krusche and Tiskin for this problem.

For the periodic IGAPS problem we give an $O(Cn^2)$ time algorithm, where C is the maximum weight of an edge. This improves upon the previous $O(C^2n^2)$ algorithm of Tiskin. We also show a reduction from periodic IGAPS to IGAPS. This reduction yields $o(n^2)$ algorithms for this problem.

1. Introduction

String comparison is a fundamental problem in computer science that has applications in computational biology, computer vision, and other areas. String comparison is often performed using *string alignment*: The characters of two input strings are aligned to each other, and a *scoring function* gives a score to the alignment according to pairs of the aligned characters and unaligned characters. The goal of the string alignment problem is to seek an alignment that maximizes (or minimizes) the score. In this paper we consider maximal scores to be optimal, but minimization problems can be solved symmetrically. The problem can be solved in $O(n^2)$ time [27], where n is the sum of lengths of A and B. Common scoring functions are the *edit distance* score, and the *LCS* (longest common subsequence) score.

A *grid graph* (see Fig. 1) is a directed graph $G = (V, E)$ whose vertex set is $V = \{(i, j)\colon\ 0 \leqslant i \leqslant m,\ 0 \leqslant j \leqslant n\}$, and whose edge set consists of three types:

1. Diagonal edges $((i, j), (i+1, j+1))$ for all $0 \leqslant i < m, 0 \leqslant j < n$.
2. Horizontal edges $((i, j), (i, j+1))$ for all $0 \leqslant i \leqslant m, 0 \leqslant j < n$.
3. Vertical edges $((i, j), (i+1, j))$ for all $0 \leqslant i < m, 0 \leqslant j \leqslant n$.

An example of a grid graph can be found in Fig. 1. In the *Grid All Paths Scores* (GAPS) problem, the input is a grid graph and the goal is to compute the maximum weights of paths between every pair of a vertex on the first row of the graph and a vertex on the last row of the graph. For simplicity of presentation, we will assume in some parts of this paper that $m = n$.

* Corresponding author.
E-mail addresses: ury@cs.bgu.ac.il (U. Matarazzo), dekelts@cs.bgu.ac.il (D. Tsur), michaluz@cs.bgu.ac.il (M. Ziv-Ukelson).

[1] The research of D.T. and U.M. was partially supported by ISF grant 981/11.
[2] The research of M.Z.U. was partially supported by ISF grant 478/10.

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2013.07.018

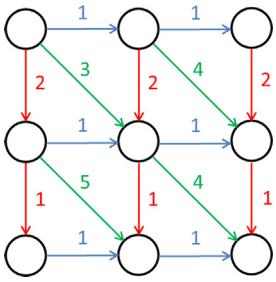

Fig. 1. A grid graph of size 2×2.

The *Integer GAPS* (IGAPS) problem is a special case of GAPS in which the weights of the edges are integers in the range 0 to C, and additionally, the weights of all the horizontal (resp., vertical) edges between two columns (resp., rows) of vertices are equal. The *Binary GAPS* (BGAPS) problem is a special case of IGAPS in which the horizontal and vertical edges have weight 0, and diagonal edges have weight 0 or 1.

The alignment problem on strings A and B can be represented by using an $(|A|+1) \times (|B|+1)$ grid graph, known as the *alignment grid graph* (cf. [21]). Vertical (respectively, horizontal) edges correspond to alignment of a character in A (respectively, B) with a gap, and diagonal edges correspond to alignment of two characters in A and B. Each edge of the graph has a weight. A path from the j-th vertex on row i to the j'-th vertex on row i' corresponds to an alignment of $A[i..i']$ and $B[j..j']$.

The GAPS problem was introduced by Apostolico et al. [3] in order to obtain fast parallel algorithms for LCS computation. It has since been studied in several additional papers [1,2,7,11–15,21–23]. Schmidt [21] showed that the GAPS problem can be solved in $O(n^2 \log n)$ time. In the same paper, Schmidt showed that IGAPS can be solved in $O(Cn^2)$ time. An $O(n^2)$ algorithm based on a similar approach for the BGAPS problem was also given by Alves et al. [1] and Tiskin [23]. Tiskin [22, p. 60] gave an $O(n^2(\log\log n/\log n)^2)$ time algorithm for a special case of BGAPS, in which the grid graph corresponds to an LCS problem on two strings. Tiskin also showed that IGAPS can be reduced to BGAPS. However, this reduction increases the size of the grid graph by a factor of C^2. Thus, the time for solving IGAPS with this reduction is either $O(C^2n^2)$ (for general grid graphs) or $O(C^2n^2(\log\log n/\log n)^2)$ (for grid graphs that correspond to alignment problems on two strings).

A special case of the BGAPS problem is when the number of diagonal edges with weight 1 is significantly smaller than n^2. We call this problem *sparse BGAPS*. Krusche and Tiskin [12] showed that sparse BGAPS can be solved in $O(n\sqrt{r})$ time, where r is the number of edges of weight 1. For the special case of a permutation grid graph (namely, each column and each row have exactly one edge of weight 1), Tiskin [22] gave an $O(n \log^2 n)$ time algorithm. Another special case of BGAPS is when the grid graph corresponds to the LCS computation of two strings with little similarity. Landau et al. [15] gave an algorithm for this variant with time complexity $O(nL)$, where L is the LCS of the two strings.

Efficient computations and storage of GAPS provide very powerful tools that can be also used for solving many problems on strings: optimal alignment computation [5], approximate tandem repeats [17,21], approximate non-overlapping repeats [4,9,21], common substring alignment [16,18], sparse spliced alignment [10,20], alignment of compressed strings [6], fully-incremental string comparison [8,19,22], and other problems.

Additional types of computations are useful in some of the applications. A *periodic grid graph* is an infinite graph obtained by concatenating horizontally an infinite number of a (finite) grid graph. The *periodic IGAPS* problem is a variant of the IGAPS problem, in which the input is a periodic grid graph. Note that while there are an infinite number of vertex pairs whose maximum path score need to be computed, due to the periodicity of the graph, the output can be represented in finite space. The periodic IGAPS problem was studied by Tiskin [25] who gave an $O(C^2n^2)$ time algorithm for the problem.

1.1. Our contribution and road map

In this work we address several variants of the IGAPS problem. Our contribution includes generalizations and improvements to previous results as follows (summarized in Table 1).

We start by working out some of the previously vague details from Schmidt's algorithm [21] for a special case of the IGAPS problem (the assumption in [21] is that all horizontal and vertical edges have weight w_1, and each diagonal edge has weight w_1 or w_2, for some fixed w_1 and w_2). We generalize Schmidt's algorithm to yield an $O(Cn^2)$ algorithm for the general IGAPS problem (Section 3).

In Section 4 we consider the sparse BGAPS problem. We give an $O(r \log^3(n^2/r))$ time algorithm, which improves the previous result of Krusche and Tiskin for this problem.

Next, we turn to address the periodic IGAPS problem in Section 5. Our first result on this front is obtained by extending the $O(Cn^2)$ algorithm for IGAPS to handle the periodic variant of the problem (Section 5.1). This improves Tiskin's $O(C^2n^2)$ result for periodic IGAPS. We then show, in Section 5.2, that periodic IGAPS can be reduced to BGAPS. Therefore, we obtain an $O(C^2n^2(\log\log n/\log n)^2)$ time algorithm for periodic IGAPS (when the grid graph corresponds to an alignment problem), and an $O(r \log^3(n^2/r))$ time algorithm for periodic sparse BGAPS.

Table 1
Results for GAPS and periodic IGAPS. The results of this paper are marked by asterisks. The results for periodic IGAPS are based on reducing the periodic problems to the non-periodic problems and using the corresponding non-periodic algorithms. The results in the third row are for the IGAPS problem when the grid graph corresponds to an alignment problem. The remaining results in the table are for general grid graphs (which may not correspond to alignment problems).

Type	Non-periodic	Periodic
IGAPS	$O(Cn^2)$ [21]	$O(C^2n^2)$ [25] $O(Cn^2)^*$
Alignment IGAPS	$O(C^2n^2(\log\log n/\log n)^2)$ [22]	$O(C^2n^2(\log\log n/\log n)^2)^*$
Sparse BGAPS	$O(n\sqrt{r})$ [12] $O(r\log^3(n^2/r))^*$	$O(r\log^3(n^2/r))^*$
Permutation	$O(n\log^2 n)$ [22]	$O(n\log^2 n)^*$

2. Preliminaries

A *sequence* is an ordered list of integers. For a sequence S, let $S[k]$ denote the k-th element of S, and let $S[k:k']$ denote the sequence $(S[k],\ldots,S[k'])$ (if $k>k'$ then $S[k:k']$ is an empty sequence). Let merge(S_1,S_2) denote the sequence obtained from merging two sorted sequences S_1 and S_2 into one sorted sequence.

Let G be an $m\times n$ grid graph with weights on the edges. Let (i,j) denote the vertex on row i and column j of the graph. The grid graph $G[i_1..i_2,j_1..j_2]$ is the subgraph obtained by taking the subgraph of G induced by the vertices $\{(i,j)\colon i_1\leqslant i\leqslant i_2, j_1\leqslant j\leqslant j_2\}$ and then renumbering the vertices by subtracting i_1 from each row number and j_1 from each column number. Let G_1 and G_2 be two grid graphs with the same number of rows. The *horizontal concatenation* of G_1 and G_2 is the grid graph obtained by merging the vertices in the last column of G_1 with the vertices of the first column of G_2 (each vertex is merged with a vertex with the same row number). The *removal of a column* j in G means taking the two subgraphs $G_1=G[0..m,0..j]$ and $G_2=G[0..m,j+1..n]$, and concatenating G_1 and G_2 horizontally. Vertical concatenation and removal of a row are defined analogously.

For a grid graph G, we will denote the weights of the diagonal, horizontal, and vertical edges leaving a vertex (i,j) by $W_{i,j}, W^H_{i,j}, W^V_{i,j}$, respectively. Recall that we assume that $W^H_{i,j}=W^H_{i',j}$ for all i,i',j, and $W^V_{i,j}=W^V_{i,j'}$ for all i,j,j'. We now claim that we can assume without loss of generality that all horizontal and vertical edges have weight 0. We show this by giving a reduction from the general case to the restricted case.[3] The first step of the reduction is to replace the weight of each diagonal edge leaving (i,j) by $W'_{i,j}=\max(W_{i,j},W^H_{i,j}+W^V_{i,j+1})$. Clearly, every path in G has the same weight under the new and original weights. The next step is to replace the weight of each diagonal edge leaving (i,j) by $W''_{i,j}=W'_{i,j}-(W^H_{i,j}+W^V_{i,j+1})$ and replace the weights of all horizontal and vertical edges by 0. It is easy to verify that the weight of a path from (i,j) to (i',j') in the original graph is equal to the weight of the path in the new graph plus $\sum_{k=j}^{j'-1}W^H_{i,k}+\sum_{k=i}^{i'-1}W^V_{k,j}$. Thus, we shall assume throughout the paper that $W^H_{i,j}=0$ and $W^V_{i,j}=0$ for all i and j.

3. Algorithm for IGAPS

Throughout this section assume that $m=n$. Define $\textsc{Opt}_i(k,j)$ to be the maximum weight of a path from $(0,k)$ to (i,j). If $k>j$ then $\textsc{Opt}_i(k,j)$ is not defined. Define

$$\textsc{DiffC}_{i,j}(k)=\textsc{Opt}_i(k,j+1)-\textsc{Opt}_i(k,j)$$

and

$$\textsc{DiffR}_{i,j}(k)=\textsc{Opt}_{i+1}(k,j)-\textsc{Opt}_i(k,j).$$

Note that $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ are defined for $0\leqslant k\leqslant j$. The $\textsc{DiffC}_{m,j}$ functions give an implicit representation of the all-scores matrix of G. Thus, our goal is to show how to compute all values of these functions.

The algorithm of Schmidt for IGAPS works as follows: It processes the input graph in top-to-bottom/left-to-right traversal order, i.e., the algorithm iterates over i from 0 to $n-1$, and for each i it iterates over j from 0 to $n-1$. For each i,j it computes all the values of $\textsc{DiffC}_{i+1,j}$ and $\textsc{DiffR}_{i,j+1}$ from the values of $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ (which were computed previously). In [21] it is claimed that the computation of $\textsc{DiffC}_{i+1,j}$ and $\textsc{DiffR}_{i,j+1}$ can be done in $O(C)$ time, without giving details. We give in Theorem 6 a complete characterization of this computation. Before giving the theorem, we give several lemmas which are either from [21] or folklore. For the sake of self-containment, we will give the proofs for these lemmas.

[3] This reduction was suggested by an anonymous referee.

Lemma 1. *For every i and j,* $\textsc{DiffC}_{i,j}(k-1) \leqslant \textsc{DiffC}_{i,j}(k)$ *and* $\textsc{DiffR}_{i,j}(k-1) \geqslant \textsc{DiffR}_{i,j}(k)$ *for all k.*

Proof. Consider some function $\textsc{DiffC}_{i,j}$ and fix some k. Let P_1 be a maximum weight path from $(0,k)$ to $(i,j+1)$, and let P_2 be a maximum weight path from $(0,k+1)$ to (i,j). The paths P_1 and P_2 must cross at some vertex v. Define x, y, z, w as follows:

- x is the weight of the prefix of P_1 until v.
- y is the weight of the prefix of P_2 until v.
- z is the weight of the suffix of P_2 from v.
- w is the weight of the suffix of P_1 from v.

Recall that $\textsc{Opt}_i(k,j)$ is the maximum weight of a path from $(0,k)$ to (i,j). The concatenation of the prefix of P_1 until v with the suffix of P_2 from v is a path from $(0,k)$ to (i,j) and consequently $x+z \leqslant \textsc{Opt}_i(k,j)$. Similarly, $y+w \leqslant \textsc{Opt}_i(k+1,j+1)$. Hence,

$$\textsc{Opt}_i(k,j+1) + \textsc{Opt}_i(k+1,j) = x+w+y+z \leqslant \textsc{Opt}_i(k,j) + \textsc{Opt}_i(k+1,j+1).$$

Therefore,

$$\textsc{DiffC}_{i,j}(k) = \textsc{Opt}_i(k,j+1) - \textsc{Opt}_i(k,j) \leqslant \textsc{Opt}_i(k+1,j+1) - \textsc{Opt}_i(k+1,j) = \textsc{DiffC}_{i,j}(k+1).$$

Since the inequality above holds for all k, it follows that $\textsc{DiffC}_{i,j}$ is monotonically non-decreasing.

The proof that $\textsc{DiffR}_{i,j}$ is monotonically non-increasing is similar. □

In the next lemma we give upper and lower bounds for $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$.

Lemma 2. $0 \leqslant \textsc{DiffC}_{i,j}(k) \leqslant C$ *and* $0 \leqslant \textsc{DiffR}_{i,j}(k) \leqslant C$ *for all* $0 \leqslant k \leqslant j$.

Proof. Every path from $(0,k)$ to (i,j) can be extended to a path from $(0,k)$ to $(i,j+1)$ by adding the vertex $(i,j+1)$ to the end of the path. The weight of the latter path is equal to the weight of the former path. Thus, $\textsc{DiffC}_{i,j}(k) \geqslant 0$.

We now prove the upper bound on $\textsc{DiffC}_{i,j}$. From every path P from $(0,k)$ to $(i,j+1)$ we can construct a path P' from $(0,k)$ to (i,j) as follows: Let v be the last vertex on P which is on column j, and let w be the next vertex on P. The path P' contains the prefix P'' of P until v and the vertices on column j from v to (i,j). Due to the assumption that all vertical edges between two rows have the same weight, the weight of P' is equal to the weight of P''. Moreover, the weight of P is equal to the weight of P'' plus the weight of the edge (v,w), which is at most C. It follows that the weight of P is less than or equal to the weight of P' plus C. Thus, $\textsc{DiffC}_{i,j}(k) \leqslant C$.

The proof of the inequalities for $\textsc{DiffR}_{i,j}$ are similar, and thus omitted. □

In the following lemmas, we will show that $\textsc{DiffC}_{i+1,j}$ and $\textsc{DiffR}_{i,j+1}$ can be computed efficiently from $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$. For every k, the values $\textsc{DiffC}_{i+1,j}(k)$ and $\textsc{DiffR}_{i,j+1}(k)$ depend on $\textsc{Opt}_{i+1}(k,j+1)$. The optimal path from $(0,k)$ to $(i+1,j+1)$ passes through either $(i+1,j)$, (i,j), or $(i,j+1)$. Thus,

$$\textsc{Opt}_{i+1}(k,j+1) = \max\big\{\textsc{Opt}_{i+1}(k,j), \textsc{Opt}_i(k,j) + W_{i,j}, \textsc{Opt}_i(k,j+1)\big\}.$$

From the equality above, we obtain the following equality for $\textsc{Opt}_{i+1}(k,j+1) - \textsc{Opt}_i(k,j)$.

Lemma 3. $\textsc{Opt}_{i+1}(k,j+1) - \textsc{Opt}_i(k,j) = \textsc{Max}_{i,j}(k)$, *where*

$$\textsc{Max}_{i,j}(k) = \max\big\{\textsc{DiffR}_{i,j}(k), W_{i,j}, \textsc{DiffC}_{i,j}(k)\big\}.$$

Proof. We have that

$$\textsc{Opt}_{i+1}(k,j+1) - \textsc{Opt}_i(k,j) = \max\left\{\begin{array}{l}\textsc{Opt}_{i+1}(k,j) - \textsc{Opt}_i(k,j),\\ \textsc{Opt}_i(k,j) - \textsc{Opt}_i(k,j) + W_{i,j},\\ \textsc{Opt}_i(k,j+1) - \textsc{Opt}_i(k,j)\end{array}\right\} = \textsc{Max}_{i,j}(k). \qquad \square$$

Lemma 4. *For* $0 \leqslant k \leqslant j$,

$$\textsc{DiffC}_{i+1,j}(k) = \textsc{Max}_{i,j}(k) - \textsc{DiffR}_{i,j}(k), \tag{1}$$

$$\textsc{DiffR}_{i,j+1}(k) = \textsc{Max}_{i,j}(k) - \textsc{DiffC}_{i,j}(k). \tag{2}$$

Proof. We have that

$$\begin{aligned}\textsc{DiffC}_{i+1,j}(k) &= \textsc{Opt}_{i+1}(k, j+1) - \textsc{Opt}_{i+1}(k, j)\\ &= \textsc{Opt}_{i+1}(k, j+1) - \textsc{Opt}_{i}(k, j) - (\textsc{Opt}_{i+1}(k, j) - \textsc{Opt}_{i}(k, j))\\ &= \textsc{Max}_{i,j}(k) - \textsc{DiffR}_{i,j}(k),\end{aligned}$$

where the last equality follows from Lemma 3. The proof of Eq. (2) is similar. □

Recall that the functions $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ were defined only for $k \leqslant j$. We now extend the definition of the $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions so that these functions will be defined for every integer k, $0 \leqslant k \leqslant n$. We want to extend each function in a way that preserves the monotonicity property and also preserves the correctness of Lemma 4. This is done by defining $\textsc{DiffC}_{i,j}(k) = C$ and $\textsc{DiffR}_{i,j}(k) = 0$ for $j < k \leqslant n$. By Lemma 2, the monotonicity of the $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions is kept.

Lemma 5. *Eqs. (1) and (2) hold for all* $0 \leqslant k \leqslant n$.

Proof. We only need to prove the lemma for $k > j$. By definition,

$$\textsc{Max}_{i,j}(k) = \max\{\textsc{DiffR}_{i,j}(k), W_{i,j}, \textsc{DiffC}_{i,j}(k)\} = \max\{0, W_{i,j}, C\} = C.$$

As we defined $\textsc{DiffR}_{i,j}(k) = 0$, we have $\textsc{Max}_{i,j}(k) - \textsc{DiffR}_{i,j}(k) = C$. We also defined $\textsc{DiffC}_{i+1,j}(k) = C$, so we conclude that Eq. (1) holds for k. Similarly, we have $\textsc{Max}_{i,j}(k) - \textsc{DiffC}_{i,j}(k) = 0$. Moreover, $\textsc{DiffR}_{i,j+1}(k) = 0$ (if $k > j+1$ this follows from definition, and if $k = j+1$ this follows from the fact that the unique path from $(0, k)$ to $(i+1, j+1)$ contains all the edges of the unique path from $(0, k)$ to $(i, j+1)$ plus the edge $((i, j+1), (i+1, j+1))$ whose weight is 0). Thus, Eq. (2) holds for k. □

The $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions are monotone functions with integer values in the range 0 to C. We define a compact representation for these functions. Intuitively, $\textsc{SC}_{i,j}$ and $\textsc{SR}_{i,j}$ are sequences that contain the "step" indices of the corresponding $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions, i.e., the indices in which the values of these sequences change. The elements of each such sequence are sorted in non-decreasing order. At this point we refer the reader to Fig. 2. Formally,

- The sequence $\textsc{SC}_{i,j}$ contains $\textsc{DiffC}_{i,j}(0)$ elements with value $-\infty$, $\textsc{DiffC}_{i,j}(k) - \textsc{DiffC}_{i,j}(k-1)$ elements with value k for every $1 \leqslant k \leqslant n$, and $C - \textsc{DiffC}_{i,j}(n)$ elements with value ∞.
- The sequence $\textsc{SR}_{i,j}$ contains $C - \textsc{DiffR}_{i,j}(0)$ elements with value $-\infty$, $\textsc{DiffR}_{i,j}(k-1) - \textsc{DiffR}_{i,j}(k)$ elements with value k for every $1 \leqslant k \leqslant n$, and $\textsc{DiffR}_{i,j}(n)$ elements with value ∞.

Each element of $\textsc{SC}_{i,j}$ and $\textsc{SR}_{i,j}$ is called a *step*. From the extended definition of the $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions it follows that each sequence $\textsc{SC}_{i,j}$ and $\textsc{SR}_{i,j}$ has exactly C elements.

Recall that by Lemma 5, $\textsc{DiffC}_{i+1,j}(k) = \textsc{Max}_{i,j}(k) - \textsc{DiffR}_{i,j}(k)$. Therefore, the steps of $\textsc{DiffC}_{i+1,j}(k)$ depend on the steps of $\textsc{Max}_{i,j}$ and $\textsc{DiffR}_{i,j}$. Based on this observation, our main result of this section shows how to compute $\textsc{SC}_{i+1,j}$ and $\textsc{SR}_{i,j+1}$ from $\textsc{SC}_{i,j}$ and $\textsc{SR}_{i,j}$. For the following theorem denote $\textsc{SC}_{i,j}[C+1] = \infty$ and $\textsc{SR}_{i,j}[0] = \infty$.

Theorem 6. *Let* $i_1 = 1 + C - W_{i,j}$ *and* $i_2 = 1 + W_{i,j}$. *If* $W_{i,j} = C$ *or* $\textsc{SR}_{i,j}[i_1 - 1] < \textsc{SC}_{i,j}[i_2]$ *then*

$$\textsc{SC}_{i+1,j} = \mathit{merge}\big(\textsc{SC}_{i,j}[i_2 : C], \textsc{SR}_{i,j}[i_1 : C]\big), \tag{3}$$

$$\textsc{SR}_{i,j+1} = \mathit{merge}\big(\textsc{SC}_{i,j}[1 : i_2 - 1], \textsc{SR}_{i,j}[1 : i_1 - 1]\big), \tag{4}$$

and otherwise

$$\textsc{SC}_{i+1,j} = S[C+1 : 2C], \tag{5}$$

$$\textsc{SR}_{i,j+1} = S[1 : C], \tag{6}$$

where $S = \mathit{merge}(\textsc{SR}_{i,j}, \textsc{SC}_{i,j})$.

Proof. We will prove the correctness of Eqs. (3) and (5). The proof for Eqs. (4) and (6) is similar.

$\textsc{SR}_{i,j}[i_1 - 1]$ is the minimum index k for which $\textsc{DiffR}_{i,j}(k) \leqslant W_{i,j}$ and $\textsc{SC}_{i,j}[i_2]$ is the minimum index k for which $\textsc{DiffC}_{i,j}(k) > W_{i,j}$ (if $\textsc{SR}_{i,j}[i_1 - 1] = -\infty$ or $\textsc{SR}_{i,j}[i_2] = -\infty$ then the corresponding k is 0). If $\textsc{SR}_{i,j}[i_1 - 1] < \textsc{SC}_{i,j}[i_2]$, then the range $\{0, \ldots, n\}$ is partitioned into three (possibly empty) regions $I_1 = \{0, \ldots, \textsc{SR}_{i,j}[i_1 - 1] - 1\}$, $I_2 = \{\textsc{SR}_{i,j}[i_1 - 1], \ldots, \textsc{SC}_{i,j}[i_2] - 1\}$, and $I_3 = \{\textsc{SC}_{i,j}[i_2], \ldots, n\}$ such that

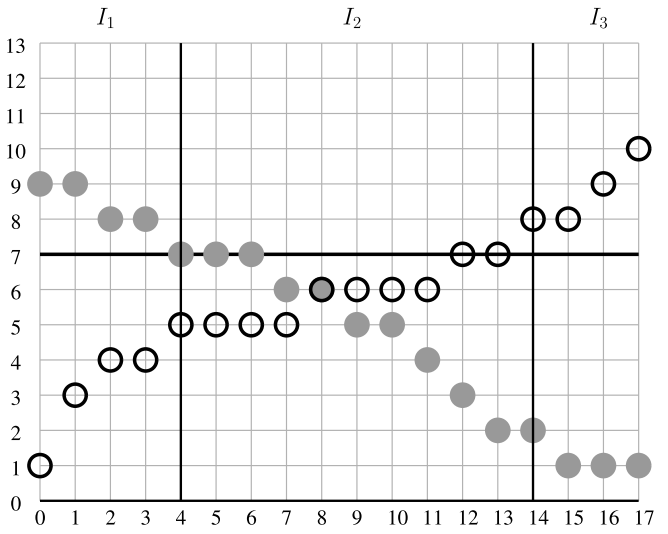

Fig. 2. An example of the case $\mathrm{SR}_{i,j}[i_1 - 1] < \mathrm{SC}_{i,j}[i_2]$. In this example, $n = 17$, $C = 10$, and $W_{i,j} = 7$. The $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$ functions are shown using black and gray, respectively. $W_{i,j}$ is marked as a horizontal black line at index 9. The steps are $\mathrm{SC}_{i,j} = (-\infty, 1, 1, 2, 4, 8, 12, 14, 16, 17)$ and $\mathrm{SR}_{i,j} = (-\infty, 2, 4, 7, 9, 11, 12, 13, 15, \infty)$. The zones are $I_1 = \{0, \ldots, 3\}$, $I_2 = \{4, \ldots, 13\}$, and $I_3 = \{14, \ldots, 17\}$. The zones are marked via vertical black lines at the corresponding zone-partitioning indices. The steps of $\textsc{DiffC}_{i+1,j}$ and $\textsc{DiffR}_{i,j+1}$ are $\mathrm{SC}_{i+1,j} = (7, 9, 11, 12, 13, 15, 14, 16, 17, \infty)$ and $\mathrm{SR}_{i,j+1} = (-\infty, -\infty, 1, 1, 2, 2, 4, 4, 8, 12)$.

$$\textsc{Max}_{i,j}(k) = \begin{cases} \textsc{DiffR}_{i,j}(k) & \text{if } k \in I_1, \\ W_{i,j} & \text{if } k \in I_2, \\ \textsc{DiffC}_{i,j}(k) & \text{if } k \in I_3. \end{cases}$$

See Fig. 2.

From Eq. (1) we obtain the following:

1. In region I_1, $\textsc{DiffC}_{i+1,j}$ is constant, so $\textsc{DiffC}_{i+1,j}$ has no steps in $I_1 \setminus \{0\}$.
2. In region I_2, $\textsc{DiffC}_{i+1,j}$ increases whenever $\textsc{DiffR}_{i,j}$ decreases, namely, the steps of $\textsc{DiffC}_{i+1,j}$ in this region are precisely the steps of $\textsc{DiffR}_{i,j}$ in this region.
3. In region I_3, $\textsc{DiffC}_{i+1,j}$ increases whenever $\textsc{DiffC}_{i,j}$ increases or $\textsc{DiffR}_{i,j}$ decreases. Thus, the steps of $\textsc{DiffC}_{i+1,j}$ in this region are the steps of $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$.

The sequences $\mathrm{SR}_{i,j}[i_1 + 1 : C]$ and $\mathrm{SC}_{i,j}[i_2 + 1 : C]$ contain the steps of $\mathrm{SR}_{i,j}$ in $I_2 \cup I_3$, and the steps of $\mathrm{SC}_{i,j}$ in I_3, respectively. Hence, the correctness of Eq. (3) follows.

If $\mathrm{SR}_{i,j}[i_1 - 1] \geqslant \mathrm{SC}_{i,j}[i_2]$ then the range $\{0, \ldots, n\}$ is partitioned into 2 regions $J_1 = \{0, \ldots, k' - 1\}$ and $J_2 = \{k', \ldots, n\}$ such that

$$\textsc{Max}_{i,j}(k) = \begin{cases} \textsc{DiffR}_{i,j}(k) & \text{if } k \in J_1, \\ \textsc{DiffC}_{i,j}(k) & \text{if } k \in J_2. \end{cases}$$

Therefore,

1. In region $J_1 \setminus \{0\}$, $\textsc{DiffC}_{i+1,j}$ has no steps.
2. In region J_2, the steps of $\textsc{DiffC}_{i+1,j}$ are the steps of $\textsc{DiffC}_{i,j}$ and $\textsc{DiffR}_{i,j}$.

The sequence $S[C + 1 : 2C]$ contains the steps of $\mathrm{SC}_{i,j}$ and $\mathrm{SR}_{i,j}$ in J_2. □

The algorithm for IGAPS follows directly from Theorem 6.

(1) **For** $j = 0, \ldots, n - 1$ **do** $\mathrm{SC}_{0,j} \leftarrow (j + 1, \ldots, j + 1)$.
(2) **For** $i = 0, \ldots, n - 1$ **do**
(3) $\quad \mathrm{SR}_{i,0} \leftarrow (-\infty, \ldots, -\infty)$.
(4) $\quad$ **For** $j = 1, \ldots, n - 1$ **do**
(5) $\quad\quad$ Compute $\mathrm{SC}_{i+1,j}$ and $\mathrm{SR}_{i,j+1}$ using Theorem 6.

By Theorem 6, the computation in line 5 takes $O(C)$ time, so the overall time complexity of the algorithm is $O(Cn^2)$.

4. Algorithm for sparse BGAPS

In what follows, an edge of weight 1 is called *active*. A row (resp., column) of G is called *inactive* if there is no active edge that starts at this row (resp., column). In this section we give an algorithm for sparse BGAPS. Our algorithm is based on the algorithm of Krusche and Tiskin [12]. Both algorithms use a divide and conquer approach, namely they divide the input grid graph into subgraphs and solve the problem recursively on each subgraph. There are two differences between these algorithms. First, the algorithm of Krusche and Tiskin stops the partitioning when a subgraph of the grid graph has no active edges, whereas our algorithm stops the partitioning when the number of active edges is at most the size of the subgraph. Furthermore, the conquer steps of the two algorithms are different. The rest of this section is organized as follows. We first describe a result of Tiskin [26] which is used for the conquer step of our algorithm. Then, we give an algorithm for handling the case when the number of active edges is at most the size of the subgraph. Finally, we describe the algorithm for sparse BGAPSand analyze its time complexity.

Due to the assumption of 0–1 weights, each sequence $\mathrm{SC}_{i,j}$ or $\mathrm{SR}_{i,j}$ contains a single element. We shall therefore refer to $\mathrm{SC}_{i,j}$ or $\mathrm{SR}_{i,j}$ as an integer rather than a sequence. Theorem 6 then reduces to the following simplified form.

Theorem 7. *(See Tiskin [24].) If* $W_{i,j} = 1$ *then* $\mathrm{SC}_{i+1,j} = \mathrm{SR}_{i,j}$ *and* $\mathrm{SR}_{i,j+1} = \mathrm{SC}_{i,j}$. *Otherwise,* $\mathrm{SC}_{i+1,j} = \max(\mathrm{SC}_{i,j}, \mathrm{SR}_{i,j})$ *and* $\mathrm{SR}_{i,j+1} = \min(\mathrm{SC}_{i,j}, \mathrm{SR}_{i,j})$.

According to the definitions in Section 2, the initialization of $\mathrm{SC}_{0,j}$ is $\mathrm{SC}_{0,j} = j + 1$. For $\mathrm{SR}_{i,0}$ we use the initialization $\mathrm{SR}_{i,0} = -i$. Note that this initialization is different from the one used in Section 2. For an $m \times n$ grid graph G we define

$$\textsc{Out}(G) = (\mathrm{SC}_{m,0}, \mathrm{SC}_{m,1}, \ldots, \mathrm{SC}_{m,n-1}, \mathrm{SR}_{m-1,n}, \mathrm{SR}_{m-2,n}, \ldots, \mathrm{SR}_{0,n}).$$

With the initialization given above and by Theorem 7, $\textsc{Out}(G)$ is a permutation of $(-m+1, -m+2, \ldots, n)$.

The algorithm of Section 3, restricted for the case of 0–1 weights, has an interpretation as a transposition network [12] (or using a different terminology, as seaweeds [22]). We now describe this interpretation using different terminology. Start with $m+n$ balls located on the edges in the first column and first row of G. The balls are numbered by $-m+1, -m+2, \ldots, n$ according to their anti-clockwise order. The balls are then moved along the edges of the graph. When the horizontal and vertical edges leaving a vertex (i, j), denoted e_1 and e_2, contain each a ball, these two balls are moved to the horizontal and vertical edges entering $(i+1, j+1)$, denoted e_3 and e_4 (the numbers of these balls represent the values $\mathrm{SC}_{i,j}$ and $\mathrm{SR}_{i,j}$, respectively). If $W_{i,j} = 1$ then the ball in e_1 is moved to e_4, and the ball in e_2 is moved to e_3. In other words, the ball in e_1 moves to the right, and the ball in e_2 is moved down. We call this movement *non-crossing*. If $W_{i,j} = 0$ then the movement of the balls depends on their numbers. If the ball in e_1 has smaller number than the ball in e_2 (indicating that the corresponding paths have already crossed once) then the movement of the ball is non-crossing. Otherwise, the movement is *crossing*: the ball in e_1 is moved to e_3 and the ball in e_2 is moved to e_4. We say in this case that the paths of the balls cross. Note that in both cases, exactly one ball is moved to e_3 and one ball is moved to e_4. Thus, for each edge on the last column and last row of the graph there is a distinct ball that reaches the edge.

When two balls reach the edges leaving some vertex and the movement for this vertex is crossing, the ball entering the horizontal edge must have a number greater than the number of the ball entering the vertical edge. If afterward these two balls reach the edges leaving another vertex, then the ball entering the horizontal edge has a number *smaller* than the number of the ball entering the vertical edge. Therefore, by definition, the movement in this step is non-crossing. In other words, the paths of two balls can cross at most once. Therefore, an alternative way to define the movement of the balls in e_1 and e_2 for the case $W_{i,j} = 0$ is: If the paths of the balls crossed before, then the movement is non-crossing, and otherwise the movement is crossing. The computation of $\textsc{Out}(G)$ is equivalent to computing the destination edges of all balls.

We will use the following results.

Theorem 8. *(See Tiskin [26].) Let G be a grid graph obtained by horizontal or vertical concatenation of two $m \times n$ grid graphs G_1 and G_2. Given* $\textsc{Out}(G_1)$ *and* $\textsc{Out}(G_2)$, $\textsc{Out}(G)$ *can be computed in* $O(n + m \log m)$ *time.*

Lemma 9. *(See Tiskin [22].) Let G be an $n \times n$ grid graph. Let G' be the grid graph obtained from G by removal of inactive rows and columns. Then* $\textsc{Out}(G)$ *can be computed from* $\textsc{Out}(G')$ *in* $O(n)$ *time.*

We now give some intuition for the term $O(n + m \log m)$ appearing in Theorem 8. The problem of computing $\textsc{Out}(G)$ from $\textsc{Out}(G_1)$ and $\textsc{Out}(G_2)$ has the following interpretation in terms of balls. The movement of balls $-m+1, \ldots, n$ in the left half of G is the same as the movement of the corresponding balls in G_1 (a ball with number i in G corresponds to the ball with number i in G_2). In particular, the n balls that reach the leftmost n horizontal edges on the last row of G are the same as the balls that reach the horizontal edges in the last row of G_1. It follows that $\textsc{Out}(G)[i][1:n] = \textsc{Out}(G_1)[i][1:n]$.

Now consider the following $m+n$ balls in G: the balls $b_{-m+1}, \ldots, b_0$ that pass through the vertical edges of column n, numbered from bottom to top, and the balls $n+1, \ldots, 2n$. Compare the movement of these balls with the movement of the balls in G_2. Here the ball with number b_i in G corresponds to the ball with number i in G_2, and the ball with number

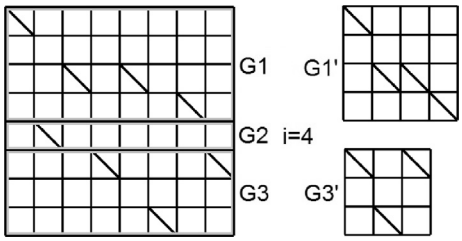

Fig. 3. An example for the algorithm for computing $\textsc{Out}(G)$ when G has at most n active edges. The figure shows the partition of G into G_1, G_2, G_3, and the new grid graphs G'_1 and G'_3 as defined in the text.

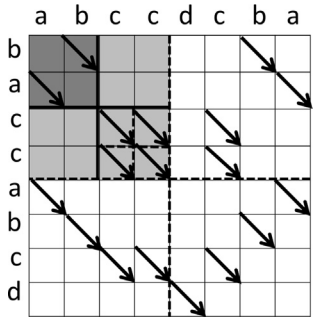

Fig. 4. An example of the algorithm for sparse BGAPS. In the first level of the recursion, the grid graph G is partitioned into four 4×4 subgraphs G_1, G_2, G_3, G_4. The number of active edges in these graphs are $6, 4, 4, 4$, respectively. Thus, the recursion stops on G_2, G_3, G_4. The subgraph G_1 is partitioned again into four subgraphs.

$n + j$ in G corresponds to the ball with number j in G_2. It is easy to verify that the movement of ball $n + j$ in G is the same as the movement of the corresponding ball j in G_2. However, the movement of a ball b_i in G may not be the same as the movement of the corresponding ball i in G_2. This is due to the fact that for two balls $b_{i_1} > b_{i_2}$ with $i_1 < i_2$, their paths crossed in the left half of G, so their paths in the right half cannot cross again. However, the paths of the corresponding balls i_1 and i_2 can cross. The crux of the algorithm of Theorem 8 is the computation the destination edges of the balls $b_{-m+1}, \ldots, b_0$ of G given the destination edges of the balls $1, \ldots, m$ of G_2. The time complexity of this computation is $O(m \log m)$, while the time of handling the rest of the balls is $O(n)$.

We now give an algorithm that computes $\textsc{Out}(G)$ for an $n \times n$ grid graph G with at most n active edges (see Fig. 3). This algorithm will later be used as a subroutine in our solution for the sparse BGAPS problem. The algorithm is an extension of an algorithm of Tiskin [22] for permutation grid graphs. Let G be a grid graph with at most n active edges. If G has at most $n/2$ active edges, then there are at least $n/2$ inactive rows and at least $n/2$ inactive columns. Choose $n/2$ inactive rows and $n/2$ inactive columns and remove them from G to obtain a grid graph G'. Recursively compute $\textsc{Out}(G')$ and then use Lemma 9 to obtain $\textsc{Out}(G)$. Otherwise, let i be the maximum index such that $G_1 = G[0..i, 0..n]$ has at most $n/2$ active edges. Note that i is well defined since $G[0..0, 0..n]$ has no active edges, and $G[0..n, 0..n]$ has more than $n/2$ active edges. Let $G_2 = G[i..i+1, 0..n]$ and $G_3 = G[i+1..n, 0..n]$. The usage of G_2 ensures that G_3 has at most $n/2$ active edges (note that G_1 also has at most $n/2$ edges by definition). Thus, we can remove $n/2$ inactive rows and $n/2$ inactive columns in each graph and obtain graphs G'_1 and G'_3. Using recursion, $\textsc{Out}(G'_1)$ and $\textsc{Out}(G'_3)$ are computed, and then $\textsc{Out}(G_1)$ and $\textsc{Out}(G_3)$ are obtained. Moreover, since $\textsc{Out}(G_2)$ is of size $1 \times n$, $\textsc{Out}(G_2)$ can be computed in $O(n)$ time. Finally, compute $\textsc{Out}(G)$ from $\textsc{Out}(G_1)$, $\textsc{Out}(G_2)$, and $\textsc{Out}(G_3)$ using Theorem 8. The time complexity function $T_2(n)$ of the algorithm satisfies the recurrence $T(n) = 2T(n/2) + O(n \log n)$. Thus, $T(n) = O(n \log^2 n)$.

Our algorithm for sparse BGAPS is as follows. Let G be an input graph of size $n \times n$. We assume that we are given as input a list of the active edges of G. For simplicity, we assume that n is a power of two.

(1) If the number of active edges is at most n, compute $\textsc{Out}(G)$ and stop.
(2) Partition G into four subgraphs $G_1 = G[0..n/2, 0..n/2]$, $G_2 = G[0..n/2, n/2..n]$, $G_3 = G[n/2..n, 0..n/2]$, and $G_4 = G[n/2..n, n/2..n]$.
(3) Recursively compute $\textsc{Out}(G_i)$ for each of the subgraphs.
(4) Compute $\textsc{Out}(G)$ by application of Theorem 8 three times.

See Fig. 4.

Time complexity analysis. Consider the total time the algorithm spends on level j of the recursion. The size of each subgraph in this level is $n' \times n'$, where $n' = n/2^j$. Clearly, the number of graphs handled in level j is at most 4^j. Moreover, the

number of subgraphs in level $j-1$ on which line (3) of the algorithm operates is at most $r/\frac{n}{2^{j-1}}$ as each such subgraph contains at least $\frac{n}{2^{j-1}}$ active edges, and these subgraphs are disjoint. It follows that the number of subgraphs in level j is at most $4\frac{r}{n/2^{j-1}}$. The time complexity of handling one subgraph in level j is either $O(n' \cdot \log^2 n')$ if step 1 is performed, and $O(n' \log n')$ if steps 2–4 are performed. Therefore, the total time of the algorithm is

$$
\begin{aligned}
O\left(\sum_{j=0}^{\log n} \min\left(4^j, \frac{r2^j}{n}\right) \cdot \frac{n}{2^j} \log^2 \frac{n}{2^j}\right) &= O\left(\sum_{j=0}^{\log \frac{r}{n}} 4^j \frac{n}{2^j} \log^2 \frac{n}{2^j} + \sum_{j=\log \frac{r}{n}+1}^{\log n} \frac{r2^j}{n} \cdot \frac{n}{2^j} \log^2 \frac{n}{2^j}\right) \\
&= O\left(n \sum_{j=0}^{\log \frac{r}{n}} 2^j \log^2 \frac{n}{2^j} + r \sum_{j=\log \frac{r}{n}+1}^{\log n} \log^2 \frac{n}{2^j}\right) \\
&= O\left(n 2^{\log \frac{r}{n}} \cdot \log^2 \frac{n}{2^{\log \frac{r}{n}}} + r\big(\log n - \log(r/n)\big) \log^2 \frac{n}{2^{\log \frac{r}{n}}}\right) \\
&= O\left(r \log^2 \frac{n^2}{r} + r \log \frac{n^2}{r} \log^2 \frac{n^2}{r}\right) \\
&= O\left(r \log^3 \frac{n^2}{r}\right).
\end{aligned}
$$

5. Algorithms for periodic IGAPS

We begin with some notations. Let G be an $n \times n$ grid graph. The *periodic grid graph* G^∞ is the graph obtained by taking an infinite number of copies of G and concatenating them horizontally. The columns of G^∞ are numbered by all (positive and negative) integers.

For the periodic IGAPS problem, we use the same notation as for the non-periodic problem, with minor differences. The $\textsc{Diffc}_{i,j}(k)$ and $\textsc{Diffr}_{i,j}(k)$ functions are defined for all integers $k \leqslant j$ and these functions are extended for all integers $k > j$ as before. The step sequences are defined as follows. $\mathrm{SC}_{i,j}$ is a sequence that contains $\min_k \textsc{Diffc}_{i,j}(k)$ elements with value $-\infty$, $\textsc{Diffc}_{i,j}(k) - \textsc{Diffc}_{i,j}(k-1)$ elements with value k for every k, and $C - \textsc{Diffc}_{i,j}(j)$ elements with value ∞. The elements of $\mathrm{SC}_{i,j}$ are sorted in non-decreasing order. The sequence $\mathrm{SR}_{i,j}$ is defined similarly. Theorem 6 also holds for the periodic problem.

Since Lemma 9 also holds for the periodic problem, we will assume without loss of generality that G does not contain inactive rows.

5.1. Direct algorithm

In this section we describe a quadratic time algorithm for periodic IGAPS. The following lemma shows that the $\mathrm{SC}_{i,j}$ and $\mathrm{SR}_{i,j}$ sequences have a periodic property. Thus, it suffices to compute $\mathrm{SC}_{i,j}$ and $\mathrm{SR}_{i,j}$ only for $0 \leqslant j \leqslant n-1$.

Lemma 10. *For all i, j, l, $\mathrm{SC}_{i,j-n}[l] = \mathrm{SC}_{i,j}[l] - n$ and $\mathrm{SR}_{i,j-n}[l] = \mathrm{SR}_{i,j}[l] - n$, where $-\infty - n = -\infty$ and $\infty - n = \infty$.*

Proof. By the periodicity of the graph, $\textsc{Diffc}_{i,j-n}(k-n) = \textsc{Diffc}_{i,j}(k)$ for all $k \leqslant j$. Therefore,

$$\textsc{Diffc}_{i,j-n}(k-n) - \textsc{Diffc}_{i,j-n}(k-n-1) = \textsc{Diffc}_{i,j}(k) - \textsc{Diffc}_{i,j}(k-1),$$

namely, the number of elements with value $k-n$ in $\mathrm{SC}_{i,j-n}$ is the same as the number of elements with this value in $\mathrm{SC}_{i,j}$. It is easy to verify that $-\infty$ and ∞ have the same number of occurrences in $\mathrm{SC}_{i,j-n}$ and $\mathrm{SC}_{i,j}$. Thus, the first part of the Lemma follows. The second part is analogous and we omit the proof. □

Similarly to the non-periodic problem, the algorithm processes a subgraph of the input graph (corresponding to $0 \leqslant j \leqslant n-1$) in top-to-bottom traversal order. The initialization of the $\mathrm{SC}_{0,j}$ sequences is the same as in the non-periodic problem (namely, $\mathrm{SC}_{0,j}$ contains C elements with value $j+1$). In the non-periodic problem, the initialization of the $\mathrm{SR}_{i,0}$ sequences is trivial (see Section 3). In the periodic problem, it may not be easy to determine $\mathrm{SR}_{i,0}$. However, the following lemma shows that in each row i there is at least one j for which the sequence $\mathrm{SR}_{i,j}$ can be easily determined.

Lemma 11. *Suppose that $W_{i,j} = \max_{j'} W_{i,j'}$. Then, for all $k \leqslant j+1$ there is a maximum weight path P from $(0,k)$ to $(i+1, j+1)$ such that the last edge in P is diagonal or vertical.*

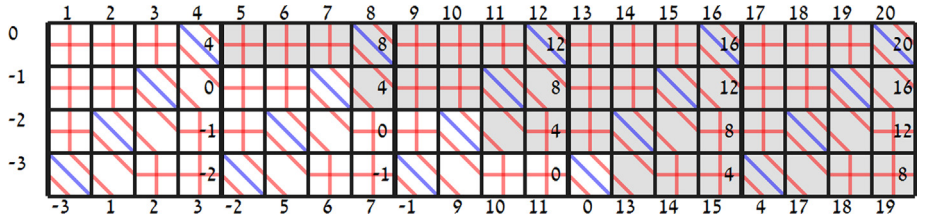

Fig. 5. The figure shows a grid graph of the form G^5, and the paths of the balls for this graph. The marked cells in figure show cells in which the entering and leaving ball numbers are the same in G^5 and in G^∞, after a shift of the values by cn for an appropriate c.

Proof. Let P be a maximum weight path from $(0,k)$ to $(i+1,j+1)$. Let (i,s) be the first vertex on P which is on row i. If $s=j+1$ then the last edge of P is the vertical edge $((i,j+1),(i+1,j+1))$ and we are done. Else, create a path P' by taking the sub-path of P from $(0,k)$ to (i,s), and joining it with the path $((i,s),(i,s+1)),\ldots,((i,j-1),(i,j)),((i,j),(i+1,j+1))$. By the assumption that $W_{i,j}=\max_{j'} W_{i,j'}$ we have that $W(P')\geqslant W(P)$ so P' is a maximum weight path from $(0,k)$ to $(i+1,j+1)$ whose last edge is diagonal. □

Lemma 12. *Suppose that $W_{i,j}=\max_{j'} W_{i,j'}$. Then, $\mathrm{SR}_{i,j+1}$ contains $C-W_{i,j}$ elements with value $-\infty$, and then the elements of $\mathrm{SC}_{i,j}[1:W_{i,j}]$.*

Proof. Fix an integer $k\leqslant j$. From Lemma 11 there is a maximum weight path P from $(0,k)$ to $(i+1,j+1)$ whose last edge is diagonal or vertical. Let P' be a path from $(0,k)$ to $(i,j+1)$ which is obtained from P by removing the last edge of P, and adding an edge $((i,j),(i,j+1))$ if this edge is not already present in P. We have that

$$\textsc{Opt}_i(k,j+1)\geqslant W\left(P'\right)=W(P)-W_{i,j}=\textsc{Opt}_{i+1}(k,j+1)-W_{i,j}.$$

We obtain that $\textsc{DiffR}_{i,j}(k)\leqslant W_{i,j}$ for all k. Therefore, the first $C-W_{i,j}$ elements of $\mathrm{SR}_{i,j}$ are $-\infty$. Thus, the case $\mathrm{SR}_{i,j}[i_1-1]<\mathrm{SC}_{i,j}[i_2]$ of Theorem 6 occurs, so

$$\mathrm{SR}_{i,j+1}=\mathrm{merge}\big(\mathrm{SC}_{i,j}[1:i_2-1],\mathrm{SR}_{i,j}[1:i_1-1]\big),$$

where $i_1=1+C-W_{i,j}$ and $i_2=1+W_{i,j}$. As mentioned above, the first $i_1-1=C-W_{i,j}$ elements of $\mathrm{SR}_{i,j}$ are all $-\infty$. Therefore, the theorem follows. □

Based on Lemma 12, the algorithm processes row i of the graph as follows. First, it finds an index j^* such that $W_{i,j^*}=\max_{j'} W_{i,j'}$, and computes SR_{i,j^*+1} according to Lemma 12. Then, it computes $\mathrm{SC}_{i+1,j}$ and $\mathrm{SR}_{i,j+1}$ for $j=j^*+1,\ldots,n-1$ using Theorem 6. Next, the algorithm computes $\mathrm{SR}_{i,0}$ from $\mathrm{SR}_{i,n}$ according to Lemma 10. Finally, it computes $\mathrm{SC}_{i+1,j}$ and $\mathrm{SR}_{i,j+1}$ for $j=0,\ldots,j^*$ using Theorem 6.

5.2. Reduction to BGAPS

In this section, we show that periodic IGAPS can be reduced to BGAPS. Since Tiskin showed a reduction from integer weights to 0–1 weights [22], it suffices to show a reduction from periodic BGAPS to BGAPS. We will use $\mathrm{SC}^G_{i,j}$ and $\mathrm{SR}^G_{i,j}$ to denote the sequences $\mathrm{SC}_{i,j}$ and $\mathrm{SR}_{i,j}$ with respect to a grid graph G.

Let G be an $n\times n$ grid graph. We extend the definition of $\textsc{Out}(\cdot)$ to periodic grid graphs as follows:

$$\textsc{Out}(G^\infty)=\left(\mathrm{SC}^{G^\infty}_{n,0},\mathrm{SC}^{G^\infty}_{n,1},\ldots,\mathrm{SC}^{G^\infty}_{n,n-1}\right).$$

To solve periodic BGAPS, it suffices to compute $\textsc{Out}(G^\infty)$.

The grid graph G^k is the graph obtained by horizontal concatenation of k copies of G (see Fig. 5). Let j_i denote the minimum index j such that $W_{i,j}=1$. The reduction from periodic IGAPS to GAPS is based on the following lemma.

Lemma 13. *Let $k\geqslant 1$. Let $0\leqslant i\leqslant n-1$ and $0\leqslant j\leqslant n-1$ be indices such that either*

1. *$i\leqslant k-1$, or*
2. *$i=k$ and $j>j_i$*

then $\mathrm{SC}^{G^k}_{i,j+\alpha}=\mathrm{SC}^{G^\infty}_{i,j}+\alpha$, where $\alpha=(k-1)n$. Moreover, for $0\leqslant i\leqslant n-1$ and $0\leqslant j\leqslant n$, if either

1. *$i\leqslant k-2$, or*
2. *$i=k-1$ and $j>j_i$*

then $\mathrm{SR}^{G^k}_{i,j+\alpha}=\mathrm{SR}^{G^\infty}_{i,j}+\alpha$.

Proof. We prove the theorem using induction on k. Fix some k. We will assume that $k > 1$ (the proof of the base of the induction, when $k = 1$, is similar). From the initialization, $\mathrm{SC}^{G^k}_{0,j+\alpha} = j+1+\alpha = \mathrm{SC}^{G^\infty}_{0,j} + \alpha$ for $0 \leqslant j \leqslant n-1$. Moreover, by the induction hypothesis we have that $\mathrm{SR}^{G^k}_{i,\alpha} = \mathrm{SR}^{G^{k-1}}_{i,n+(k-2)n} = \mathrm{SR}^{G^\infty}_{i,n} + (k-2)n$ for $0 \leqslant i \leqslant k-2$. By Lemma 10, $\mathrm{SR}^{G^\infty}_{i,n} = \mathrm{SR}^{G^\infty}_{i,0} + n$. It follows that $\mathrm{SR}^{G^k}_{i,\alpha} = \mathrm{SR}^{G^\infty}_{i,0} + \alpha$ for $0 \leqslant i \leqslant k-2$.

From Theorem 7 we obtain the following corollary: If $\mathrm{SC}^{G^k}_{i,j+\alpha} = \mathrm{SC}^{G^\infty}_{i,j} + \alpha$ and $\mathrm{SR}^{G^k}_{i,j+\alpha} = \mathrm{SR}^{G^\infty}_{i,j} + \alpha$ then $\mathrm{SC}^{G^k}_{i+1,j+\alpha} = \mathrm{SC}^{G^\infty}_{i+1,j} + \alpha$ and $\mathrm{SR}^{G^k}_{i,j+1+\alpha} = \mathrm{SR}^{G^\infty}_{i,j+1} + \alpha$ (this is due to the fact that the diagonal edge leaving $(i, j+\alpha)$ in G^k has the same weight as the diagonal edge leaving (i, j) in G^∞). We now apply the corollary on $i = 0$ and $j = 0$ as we have already shown above that the conditions of the corollary are satisfied. We next apply the corollary on $i = 0$ and $j = 1$ (the fact that the condition $\mathrm{SR}^{G^k}_{i,j+\alpha} = \mathrm{SR}^{G^\infty}_{i,j} + \alpha$ is satisfied follows from the application of the corollary on $i = 0$ and $j = 0$). We can continue applying the corollary for $i = 0$ and $j = 2, 3, \ldots, n-1$. Next, we apply the corollary for $i = 1$ and $j = 0, 1, \ldots, n-1$, and then we continue applying the corollary for all $i \leqslant k-2$ and $j \leqslant n-1$.

Now, consider the indices $i = k-1$ and $j = j_i$. From above, we have that $\mathrm{SC}^{G^k}_{i,j+\alpha} = \mathrm{SC}^{G^\infty}_{i,j} + \alpha$. Since by definition, the diagonal edges leaving $(i, j+\alpha)$ in G^k and (i, j) in G^∞ have weight 1, we obtain from Theorem 7 that $\mathrm{SR}^{G^k}_{i,j+\alpha+1} = \mathrm{SC}^{G^k}_{i,j+\alpha} = \mathrm{SC}^{G^\infty}_{i,j} + \alpha = \mathrm{SR}^{G^\infty}_{i,j+1} + \alpha$. We can now apply the corollary above on $i = k-1$ and $j = j_i + 1$, and continue applying the corollary for $i = k-1$ and $j = j_i + 2, \ldots, n-1$. □

Corollary 1. *If $k \geqslant n+1$ then* $\textsc{Out}(G^\infty) = (\mathrm{SC}^{G^k}_{n,\alpha} - \alpha, \mathrm{SC}^{G^k}_{n,1+\alpha} - \alpha, \ldots, \mathrm{SC}^{G^k}_{n,n-1+\alpha} - \alpha)$.

Our goal is to show how to compute $(\mathrm{SC}^{G^k}_{n,\alpha}, \mathrm{SC}^{G^k}_{n,1+\alpha}, \ldots, \mathrm{SC}^{G^k}_{n,n-1+\alpha})$ for some $k \geqslant n+1$. Define

$$\textsc{Out}'(G^k) = \left(\mathrm{SC}^{G^k}_{n,\alpha}, \mathrm{SC}^{G^k}_{n,1+\alpha}, \ldots, \mathrm{SC}^{G^k}_{n,n-1+\alpha}, \mathrm{SR}^{G^k}_{n-1,kn}, \mathrm{SR}^{G^k}_{n-2,kn}, \ldots, \mathrm{SR}^{G^k}_{0,kn}\right).$$

The following lemma follows from Theorem 8.

Lemma 14. *Given* $\textsc{Out}'(G^k)$, $\textsc{Out}'(G^{2k})$ *can be computed in* $O(n \log n)$ *time.*

Based on Corollary 1 and Lemma 14, we obtain the following algorithm for solving periodic BGAPS.

(1) Compute $\textsc{Out}'(G)$.
(2) Let n' be the smallest power of 2 which is greater than or equal to n.
(3) **For** $k = 1, 2, 4, \ldots, n'/2$ **do**
(4) Compute $\textsc{Out}'(G^{2k})$ from $\textsc{Out}'(G^k)$.
(5) Output $(\textsc{Out}'(G^{n'})[1] - (n'-1)n, \ldots, \textsc{Out}'(G^{n'})[n] - (n'-1)n)$.

We have shown the following theorem.

Theorem 15. *The periodic BGAPS problem on a grid graph G can be solved in $T(G) + O(n \log^2 n)$ time, where $T(G)$ is the time complexity of solving BGAPS on G.*

References

[1] C.E.R. Alves, E.N. Cáceres, S.W. Song, An all-substrings common subsequence algorithm, Discrete Applied Mathematics 156 (7) (2008) 1025–1035.
[2] A. Apostolico, M.J. Atallah, S.E. Hambrusch, New clique and independent set algorithms for circle graphs, Discrete Applied Mathematics 36 (1) (1992) 1–24.
[3] A. Apostolico, M.J. Atallah, L.L. Larmore, S. McFaddin, Efficient parallel algorithms for string editing and related problems, SIAM Journal on Computing 19 (5) (1990) 968–988.
[4] G. Benson, A space efficient algorithm for finding the best nonoverlapping alignment score, Theoretical Computer Science 145 (1&2) (1995) 357–369.
[5] M. Crochemore, G.M. Landau, M. Ziv-Ukelson, A sub-quadratic sequence alignment algorithm for unrestricted cost matrices, SIAM Journal on Computing 32 (5) (2003) 1654–1673.
[6] D. Hermelin, G.M. Landau, S. Landau, O. Weimann, A unified algorithm for accelerating edit-distance computation via text-compression, in: Proc. 26th Symposium on Theoretical Aspects of Computer Science, STACS, 2009, pp. 529–540.
[7] H. Hyyrö, An efficient linear space algorithm for consecutive suffix alignment under edit distance, in: Proc. 15th Symposium on String Processing and Information Retrieval, SPIRE, 2008, pp. 155–163.
[8] Y. Ishida, S. Inenaga, A. Shinohara, M. Takeda, Fully incremental LCS computation, in: Proc. 15th Symposium on Fundamentals of Computation Theory, FCT, 2005, pp. 563–574.
[9] S. Kannan, E.W. Myers, An algorithm for locating nonoverlapping regions of maximum alignment score, SIAM Journal on Computing 25 (3) (1996) 648–662.
[10] C. Kent, G.M. Landau, M. Ziv-Ukelson, On the complexity of sparse exon assembly, Journal of Computational Biology 13 (5) (2006) 1013–1027.
[11] S.-R. Kim, K. Park, A dynamic edit distance table, Journal of Discrete Algorithms 2 (2) (2004) 303–312.
[12] P. Krusche, A. Tiskin, String comparison by transposition networks, in: Proc. of London Algorithmics Workshop, 2008, pp. 184–204.

[13] P. Krusche, A. Tiskin, New algorithms for efficient parallel string comparison, in: Proc. 22nd Symposium on Parallel Algorithms and Architectures, SPAA, 2010, pp. 209–216.
[14] G.M. Landau, E.W. Myers, J.P. Schmidt, Incremental string comparison, SIAM Journal on Computing 27 (2) (1998) 557–582.
[15] G.M. Landau, E.W. Myers, M. Ziv-Ukelson, Two algorithms for LCS consecutive suffix alignment, Journal of Computer and System Sciences 73 (7) (2007) 1095–1117.
[16] G.M. Landau, B. Schieber, M. Ziv-Ukelson, Sparse LCS common substring alignment, Information Processing Letters 88 (6) (2003) 259–270.
[17] G.M. Landau, J.P. Schmidt, D. Sokol, An algorithm for approximate tandem repeats, Journal of Computational Biology 8 (1) (2001) 1–18.
[18] G.M. Landau, M. Ziv-Ukelson, On the common substring alignment problem, Journal of Algorithms 41 (2) (2001) 338–359.
[19] L.M.S. Russo, Multiplication algorithms for Monge matrices, in: Proc. 17th Symposium on String Processing and Information Retrieval, SPIRE, 2010, pp. 94–105.
[20] Y. Sakai, An almost quadratic time algorithm for sparse spliced alignment, Theory of Computing Systems (2009) 1–22.
[21] J.P. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, SIAM Journal on Computing 27 (4) (1998) 972–992.
[22] A. Tiskin, Semi-local string comparison: algorithmic techniques and applications, arXiv:0707.3619v16.
[23] A. Tiskin, Semi-local longest common subsequences in subquadratic time, Journal of Discrete Algorithms 6 (4) (2008) 570–581.
[24] A. Tiskin, Semi-local string comparison: Algorithmic techniques and applications, Mathematics in Computer Science 1 (4) (2008) 571–603.
[25] A. Tiskin, Periodic string comparison, in: Proc. 20th Symposium on Combinatorial Pattern Matching, CPM, 2009, pp. 193–206.
[26] A. Tiskin, Fast distance multiplication of unit-Monge matrices, in: Proc. 21st Symposium on Discrete Algorithms, SODA, 2010, pp. 1287–1296.
[27] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, Journal of the ACM 21 (1) (1974) 168–173.