From de Bruijn Graphs to Rectangle Graphs for Genome Assembly

Nikolay Vyahhi[1], Alex Pyshkin[1], Son Pham[2,⋆], and Pavel A. Pevzner[1,2]

[1] Algorithmic Biology Laboratory, St. Petersburg Academic University, Russia
[2] Department of Computer Science and Engineering, UCSD, La Jolla, CA, USA
kspham@cs.ucsd.edu

Abstract. Jigsaw puzzles were originally constructed by painting a picture on a rectangular piece of wood and further cutting it into smaller pieces with a jigsaw. The *Jigsaw Puzzle Problem* is to find an arrangement of these pieces that fills up the rectangle in such a way that neighboring pieces have "matching" boundaries with respect to color and texture. While the general Jigsaw Puzzle Problem is NP-complete [6], we discuss its simpler version (called *Rectangle Puzzle Problem*) and study the *rectangle graphs*, recently introduced by Bankevich et al., 2012 [3], for assembling such puzzles. We establish the connection between Rectangle Puzzle Problem and the problem of assembling genomes from read-pairs, and further extend the analysis in [3] to real challenges encountered in applications of rectangle graphs in genome assembly. We demonstrate that addressing these challenges results in an assembler SPAdes+ that improves on existing assembly algorithms in the case of bacterial genomes (including particularly difficult case of genome assemblies from single cells).

SPAdes+ is freely available from http://bioinf.spbau.ru/spades.

1 Introduction

The recent proliferation of next generation sequencing technologies has enabled new experimental opportunities and, at the same time, raised formidable computational challenges. When the length of a repeat in the genome exceeds the read length, it becomes difficult to "span" the flanking regions of this repeat in the assembly. To alleviate this problem, sequencing technologies were extended to produce *read-pairs*, pairs of reads separated by an estimated *insert length*. Because insert length is longer than the read length, read-pairs span longer repeats and could potentially result in better assemblies. However, while assembling *single* reads can be elegantly modeled by *de Bruijn graphs* [7], equally elegant models for assembling *read-pairs* remain unknown [6].

Pevzner and Tang, 2001 [11] addressed this challenge by constructing the de Bruijn graph and further checking if a path between two reads within a read-pair satisfies the constraint imposed by the insert length. If only one such path exists,

⋆ Corresponding author.

B. Raphael and J. Tang (Eds.): WABI 2012, LNBI 7534, pp. 249–261, 2012.

the read-pair is transformed into a virtual long read where the gap between reads is filled in with the nucleotide sequence representing the found path.

While this and similar methods [9, 13] had a large impact on genome assembly, they fail in repeat-rich regions, where there are multiple paths between the reads within read-pairs. Recently, Medvedev et al., 2011 [10] introduced *paired de Bruijn graphs* that directly incorporate read-pairs into the graph structure and bypass the problem of multiple paths in previous approaches. However, the paired de Bruijn graph concept was introduced as a theoretical framework and is mainly aimed at an unrealistic case when the distance between reads within read-pairs is exactly d for all read-pairs. To address this bottleneck, Bankevich et al., 2012 [3] introduced the *rectangle graph* by generalizing the problem of a string reconstruction from its paired substrings to a variation of a jigsaw puzzle problem. While fragment assembly is usually modeled as a 1-dimensional *overlapping* puzzle (pieces correspond to individual reads), Bankevich et al. [3] modeled assembly as a 2-dimensional *non-overlapping* puzzle (pieces correspond to pairs of paths in the de Bruijn graph). However, while Bankevich et al. [3] sketched the rectangle graph idea, the various questions arising in applications of rectangle graphs to fragment assembly remained unaddressed.

The jigsaw puzzles were originally constructed by painting a picture on a rectangular piece of wood and further cutting it into smaller pieces with a jigsaw. The *Jigsaw Puzzle Problem* is to find an arrangement of these pieces that fills up the rectangle in such a way that neighboring pieces have "matching" boundaries with respect to color and texture. This paper extends the previous algorithmic studies of the Jigsaw Puzzle Problem (that were motivated by the restoration of archaeological artifacts [8]) to the problem of genome assembly from read-pairs. In section 2, we describe a class of simple jigsaw puzzles (called *rectangle puzzles*) and define the rectangle graph to assemble such puzzles. In section 3, we establish the relation between the rectangle puzzle and genome assembly from read-pairs and address the algorithmic challenges of "missing rectangles" (that often arise in genome assembly) in the rectangle puzzle problem. In Section 4, we apply the rectangle graph to bacterial genome assembly for both standard (multicell) and more difficult single cell datasets.

2 Rectangle Puzzles

Consider $n+1$ points $x_0 = 0 < x_1 < \ldots < x_n$ on x-axis and $m+1$ points $y_0 = 0 < y_1 < \ldots < y_m$ on y-axis. Points (x_i, y_j) form a 2-dimensional grid consisting of $n \cdot m$ rectangles filling up the grid with corners $(0,0), (0,y_m), (x_n,0)$ and (x_n, y_m). By analogy with the jigsaw puzzle assembly, we assume that the grid is "painted" and the goal is to assemble small rectangles into the painted grid in such a way that rectangles fully fill up the grid and that the colors at the sides of neighboring rectangles match (*valid assembly*). To simplify the matters, we will assume that the "orientation" of each rectangle is known.

Assembling the Ha Long Bay puzzle in Fig. 1a is trivial (for every rectangle, there exists an unambiguous choice of neighboring rectangles). Fig. 1b shows 9 rectangles from this puzzle with 12 blue dotted lines connecting the *unique*

matching sides of these rectangles. Fig. 1c shows a more difficult "frogs and butterflies" puzzle (for every rectangle, there are *multiple* choices of neighboring rectangles). Fig. 1d shows 9 rectangles from this puzzle with 6 dotted connections showing all rectangles that match the upper side of the lower-left rectangle. Even a seasoned puzzle enthusiast may have difficult time assembling this puzzle and may end up with a wrong assembly shown in Fig. 1e [1]. Below we introduce a simpler type of puzzles (shown in Fig. 1f and called *rectangle puzzles with traversing curves*) and discuss algorithms for their assembly.

Consider a continuous non-self-intersecting curve from $(0,0)$ to (x_n, y_m) in the grid that crosses the sides of the rectangles at points $p_0 = (0,0), p_1, \ldots, p_N = (x_n, y_m)$ (in their order along the curve). For convenience, we assume that points on this curve are painted "red" and no other points in the puzzle is painted red. The curve is called *traversing* [2] if p_i and p_{i+1} belong to different sides of a rectangle (for $0 \leq i \leq N-1$) and if there are exactly two red points on the sides of each rectangle. For simplicity we assume that the direction of the traversing curve within each rectangle is known and that the curve does not pass through the corners of rectangles except for the points $p_0 = (0,0)$ and $p_N = (x_n, y_m)$.

In the Rectangle Puzzle Problem we assume that red points in the grid form a traversing curve and the goal is to assemble the grid from rectangles. More precisely, we want to generate all valid assemblies of the rectangles into the grid. Fig. 1g shows 9 rectangles from this puzzle with blue dotted lines connecting all matching sides of these rectangles and illustrates that the number of matching sides is reduced as compared to Fig 1d.

The red curve enters and leaves each rectangle R through sides that we call $source(R)$ and $sink(R)$ correspondingly. We assume that every side S of a rectangle is assigned a label $label(S)$ (that encodes the painting of this side) and that differently painted sides are assigned different labels. We represent a rectangle R by a directed edge $edge(R)$ from vertex $label(source(R))$ to vertex $label(sink(R))$. For convenience, we assign identical and unique labels to the sides of rectangles containing the first and the last points $(0,0)$ and (x_n, y_m) on the red curve. It corresponds to closing the traversing curve as shown in Fig. 1g.

The concept of the traversing curve turns out to be useful for bringing the de Bruijn graph concept in the domain of puzzle assembly. Below, we describe an application of this concept for assembling rectangle puzzles.

Rectangle Graphs. The concept of rectangle graphs was first described in [3]. Given a rectangle puzzle, its *rectangle graph* is constructed as follows:

- Define a graph G with $n \cdot m$ isolated edges (on $2 \cdot n \cdot m$ vertices) by introducing a directed edge $edge(R)$ for each rectangle R. Starting and ending vertices of $edge(R)$ are labeled as $label(source(R))$ and $label(sink(R))$, correspondingly.
- The rectangle graph is formed by gluing identically labeled vertices in G (Fig. 1h).

[1] Polynomial algorithms for assembling such puzzles remain unknown.

[2] Intuitively, a traversing curve is a curve that "visits" every rectangle exactly once.

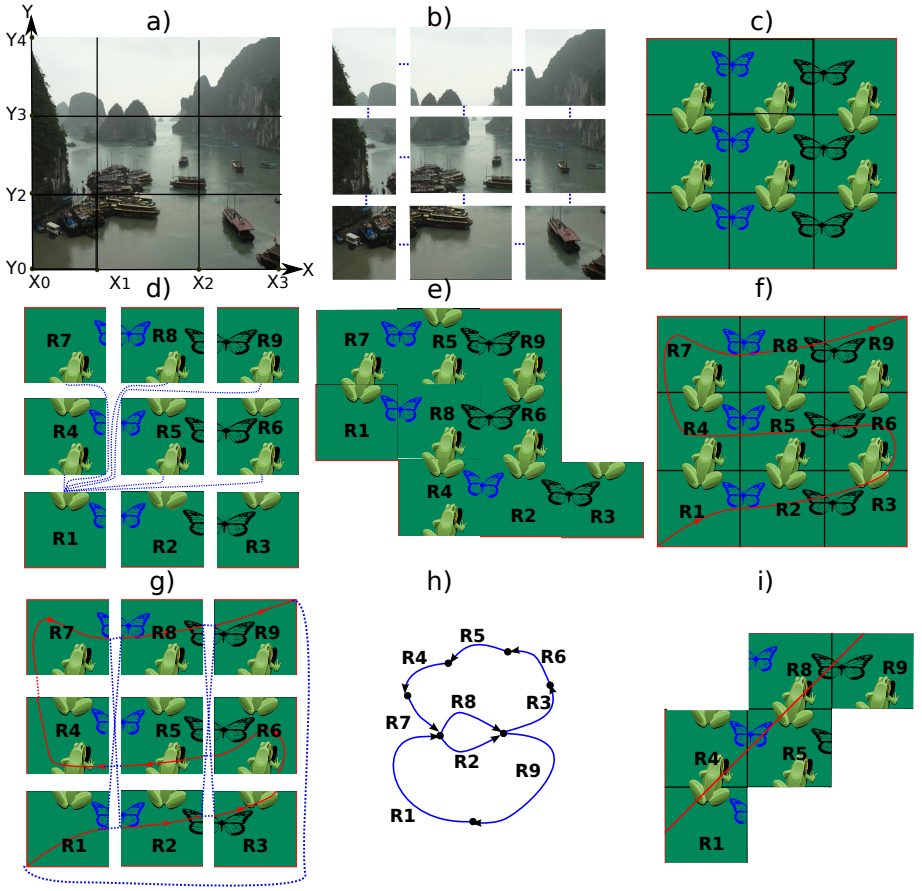

Fig. 1. Rectangle puzzles and rectangle graphs. a) A simple puzzle on the background image of Ha Long Bay. Points $x_0, \ldots, x_3$ on x-axis and $y_0, \ldots, y_3$ on y-axis form a 3×3 grid. (b) Nine rectangles with twelve matching sides in the Ha Long bay puzzle (shown by blue dotted connections) illustrate that there exists an unambiguous choice of matching sides. (c) A more difficult "frogs and butterflies" puzzle with multiple ambiguous choices of matching sides. (d) Nine rectangles with multiple matching sides (only some of them are shown) illustrate ambiguities in the selection of matching sides. (e) Failed attempt at the "frogs and butterflies" puzzle assembly. (f) The traversing curve in the "frogs and butterflies" puzzle makes the assembly easier. (g) Gluing sides of the rectangles in the "frogs and butterflies" puzzle. (h) The rectangle graph with 2 Eulerian cycles: $R_1R_2R_3R_6R_5R_4R_7R_8R_9$ and $R_1R_8R_3R_6R_5R_4R_7R_2R_9$ where only the first represents a valid solution. (i) Traversing line and subgrid assembly. The red traversing curve is replaced by a line. We are interested in assembling the subgrid formed by all rectangles crossed by the red line.

Obviously, the rectangle graph is the de Bruijn graph on strings of length 2 in the alphabet of labels (each label encodes a side of a rectangle). Each rectangle assembly corresponds to an Eulerian cycle in the rectangle graph. All Eulerian cycles can be generated using the BEST [1] theorem thus reducing the rectangle puzzle assembly to enumerating Eulerian cycles in the rectangle graph [2]. However, not every Eulerian cycle corresponds to a valid solution of the rectangle puzzle since some solutions may correspond to: (i) an assembly where rectangles overlap, (ii) an assembly that does not form a rectangular grid, (iii) an assembly where some sides of rectangles do not match. While the number of Eulerian cycles may be large, it is easy to check if a given Eulerian cycle corresponds to a valid rectangle puzzle assembly in linear time.

Below we limit attention to traversing *lines* (rather than *curves*) and relax the condition of *visiting* all rectangles (Fig. 1i): The traversing line $y = x + d$ visits *some* (not necessary all) rectangles. In this case we are only interested in assembling rectangles into a *subgrid* formed by rectangles crossed by the red line, rather than assembling all rectangles into the full grid. For $d \neq 0$, the traversing line does not necessarily starts at $(0, 0)$ or ends at (x_n, y_m). In this case, every Eulerian cycle corresponds to a valid subgrid assembly. It is easy to see that in the case of the traversing *line* (in difference from the traversing *curve*) no additional checks are needed to verify that the assembly (given by an Eulerian cycle) is valid. Below we continue using the term "rectangle puzzle" while referring to the case of traversing *lines* (rather than traversing *curves*).

3 Rectangle Puzzles and Genome Assembly

Generating a Rectangle Puzzle from a Genome

We represent a genome as a circular string over the alphabet of nucleotides $\{A, T, C, G\}$. A *k-mer* is a string of length k in the alphabet of nucleotides.

Given a k-mer $s = s_1 \dots s_k$, we define $prefix(s) = s_1 \dots s_{k-1}$ and $suffix(s) = s_2 \dots s_k$. Given a *Genome*, the de Bruijn graph $DB(Genome, k)$ is defined on the set of vertices representing all $(k-1)$-mers from *Genome* and has a directed edge $(prefix(s), suffix(s))$ for each k-mer s appearing in *Genome*. It is easy to see that *Genome* defines an Eulerian cycle in its de Bruijn graph.

A vertex v in a graph precedes (follows) a vertex w if there exists an edge from v to w (from w to v). The indegree (outdegree) of a vertex is the number of vertices preceding (following) it. A vertex is called a branching vertex if either its indegree or its outdegree is larger than 1. A path in a graph is called a *non-branching path* if all vertices in this path (with exception of the first and the last ones) have indegree and outdegree both equal to 1.

The de Bruijn graph $DB(Genome, k)$ partitions $(k-1)$-mers from *Genome* into branching (if they correspond to branching vertices in $DB(Genome, k)$) and non-branching. Similarly, all positions in *Genome* are partitioned into branching (if the $(k-1)$-mer starting at this position is branching) and non-branching. For example, ACG, CGT, GTT, and TCT are the branching 3-mers in *Genome*

(shown as red points in Fig. 2a) while 0, 1, 7, 9, 13, 14, 19, 21, 24 are branching positions (shown as red points in Fig. 2b). For convenience, we assume that the circular genome "starts" at a branching position 0 and "ends" at the branching position N.[3]

We denote the branching positions in *Genome* as $x_0 = 0 < x_1 < \ldots x_n = N$ and define the grid consisting of $n \cdot n$ rectangles formed by points $x_0 = 0 < x_1 < \ldots < x_n = N$ on x axis and the same list of points on y-axis. The segment of *Genome* between positions x_i and x_{i+1} corresponds to a non-branching path in the de Bruijn graph. Thus, every rectangle corresponds to a pair of non-branching paths. The red line is defined by the equation $y = x + d$ [4]. Given an integer d, we define $Puzzle(Genome, k, d)$ as a set of rectangles crossed by the red line. Each rectangle in this set is uniquely defined by a pair of non-branching paths and the position of a red line segment within the rectangle.

Given a position x in *Genome* we define $(\bar{x})$ as the $(k-1)$-mer starting at this position. Thus, each integer 2D coordinate (x, y) defines a *paired* $(k-1)$-*mer* $(a|b)$, where $a = (\bar{x})$ and $b = (\bar{y})$. The label ("paint") of position (x, y) in the grid is defined as $((\bar{x}), (\bar{y}), color)$. where *color* is "red" or "white" depending on whether (x, y) is located on the red line or not. The label of a side of a rectangle is defined as an ordered list of all labels of (integer) points located on this side. It is easy to see that there exists an alternative simpler representation of this label, i.e., by a paired $(k-1)$-mer corresponding to the red position on the corresponding side [5]. A rectangle formed by a pair of non-branching paths (p, p') together with the red line segment on it can be represented as a triple (p, p', t) where t is the position of the red line in the rectangle relative to the low left corner of the rectangle. See Fig. 2b for an example of rectangle puzzle constructed from a genome.

We mention that since the labels along the red line completely define the genome, assembling the red line from the rectangles results in assembling the genome. However, this puzzle may appear useless for genome assembly tasks since the puzzle itself was originally created from the genome that we are trying to assemble in the first place! Below we show that the rectangle puzzle can be created from read-pairs without knowing the genome.

Generating a Rectangle Puzzle from Exact-Distance Read-Pairs
A (k, d)-mer is a pair of k-mers separated by d in the genome. If two reads $r'_1 \ldots r'_n$ and $r''_1 \ldots r''_n$ within a read-pair are separated by an exact distance d, one can extract (k, d)-mers $(r'_i \ldots r'_{i+k-1} | r''_i \ldots r''_{i+k-1})$ from them (for $1 \le i \le n - k + 1$). Iterating over all read-pairs results in a large set of (k, d)-mers. For simplicity, we unrealistically assume that the resulting set contains all and only (k, d)-mers from the genome. Before showing that the $Puzzle(Genome, k, d)$ can be constructed only from the (k, d)-mers set without knowing the genome, we

[3] Since the genome is circular, these two positions represent the same site in the genome and the same vertex in the de Bruijn graph.
[4] Below we will define d as the median distance between reads within a read-pair.
[5] Since it uniquely defines the label of all other points on the side.

introduce *multirectangle* — a different but equivalent representation of rectangle pieces in the rectangle puzzle [6] (Fig. 3a).

Given r rectangles: $(p, p', t_1), \ldots, (p, p', t_r)$ formed by the same pair of non-branching paths (p, p') but having different positions of their red line segments, we define a multirectangle R^* as a rectangle that is formed by the same pair of non-branching paths (p, p') (with horizontal edges p and vertical edges p') but with r red line segments and represent the multirectangle as $(p, p', \{t_1, \ldots, t_r\})$.

Let $Puzzle^*(Genome, k, d)$ denote a set of multirectangles by transforming rectangles in $Puzzle(Genome, k, d)$ that are formed by the same pairs of non-branching paths into single multirectangles. Within each multirectangle $R^* = (p, p', \{t_1, \ldots t_r\}) \in Puzzle^*(Genome, k, d)$, the red (integer) points on these r line segments represent *all* $(k-1, d)$-mers $(a|b)$ of the genomes such that $a \in p$ and $b \in p'$ [7]. Additionally, each $(k-1, d)$-mer $(a|b)$ such that $a \in p$ and $b \in p'$, corresponds to a *unique* position in the multirectangle. These two observations lead to a simple approach for constructing the $Puzzle^*(Genome, k, d)$ from the (k, d)-mers set: (1) Construct the de Bruijn graph $DB(ReadPairs, k)$ from individual reads in read-pairs; (2) For each pair of non-branching paths (p, p') in the de Bruijn graph that are *connected* by (k, d)-mers (i.e., there exists at least one $(k-1, d)$-mer $(a|b)$ such that $a \in p$ and $b \in p'$), we draw a multirectangle with horizontal edges p and vertical edges p', together with red points corresponding to $(k-1, d)$-mer that connect p and p'. These points form a single or multiple red line segments within the multirectangle.

From the set of multirectangles, we further transform it into the rectangle puzzle by replacing each multirectangle by separated rectangles, each containing a *single* red line segment (see Fig. 3a).

Generating a Rectangle Puzzle from Inexact-Distance Read-Pairs
We now show how to construct the rectangle puzzle in a more realistic case of read-pairs with inexact distances between reads. Given integers d and Δ,[8] a pair of k-mers $(a|b)$ is called a (k, d, Δ)-mer in $Genome$ if it is a (k, d_0)-mer of $Genome$ for some $d_0 \in [d-\Delta, d+\Delta]$. While the set of all $(k-1, d)$-mer of $Genome$ forms a line $(d) : y = x + d$ in the 2D grid, a set of $(k-1, d, \Delta)$-mers fills up a band of width Δ around (d), called a *Δ-cloud*. In this case, the rectangle puzzle needs to be redefined since: (1) red line segments in rectangles are substituted by red Δ-clouds, making it difficult to infer the position of the red line segments within the rectangles; (2) some new rectangles with red points are added into the original set of rectangles crossed by the red line (false rectangles); (3) some rectangles crossed by the red line are now missing (missing rectangles). Below we address these complications.

[6] While introducing the multirectangle concept does not have any analogy to the jigsaw puzzle assembly, it simplifies the proof that the $Puzzle(Genome, k, d)$ can be constructed only from the set of all (k, d)-mers of the unknown genome.

[7] With a minor exception for points that lie on the edges of the multirectangles.

[8] Integer d refers to the median distance between reads within a read-pair while integer Δ refers to the maximum deviation of the distance between the reads within a read-pair.

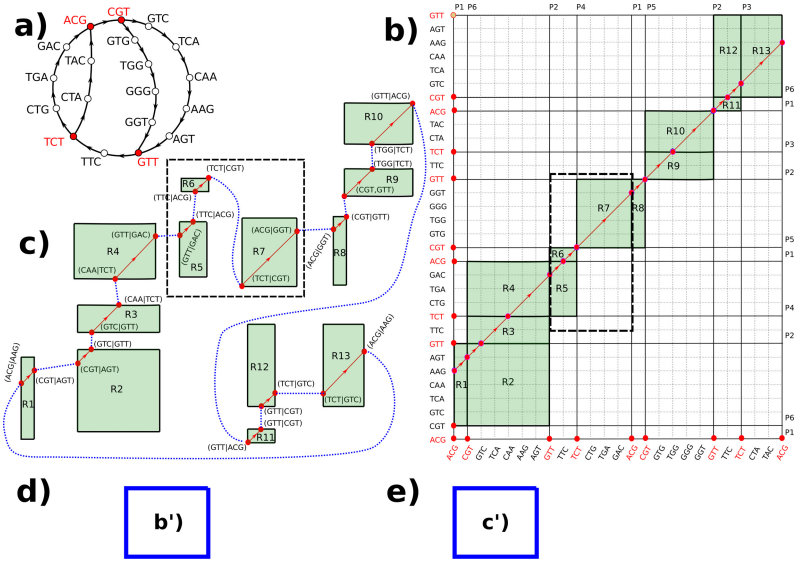

Fig. 2. The rectangle puzzle and the rectangle graph of the genome $Genome = ACGTCAAGTTCTGACGTGGGTTCT$. (a) De Bruijn graph $DB(Genome, k)$ with $k = 4$ can be constructed from $Genome$ or individual reads generated from $Genome$. The graph has 4 branching vertices (ACG, CGT, GTT, TCT) colored red that correspond to 8 branching positions in $Genome$. (b) Generating rectangle puzzle when $Genome$ is known. $Genome$ is represented as a sequence of 3-mers in both vertical and horizontal axes. The 3-mers corresponding to the branching vertices in the de Bruijn graph are colored red. The set of all $(3, 5)$-mers (pair of 3-mers separated by 5 nucleotides in the genome) forms a line $(d) : y = x + 5$ on the grid. (c) Rectangle graph is obtained by gluing sides of rectangle with the same labels. (d) The same as figure (b) but with rectangles in the dash box (R_5, R_6, R_7) removed. R_5, R_6, R_7 represent 3 missing rectangles. (e) The same as figure (c) but with rectangles in the dash box (R_5, R_6, R_7) missing. This results in two dead-ends vertices (sides of R_4 and R_8) in the rectangle graph.

For each pair of non-branching path (p, p') in the de Bruijn graph that is connected by $(k-1, d, \Delta)$-mers, we form a *multirectangle* with horizontal edge p and vertical edge p' together with red points corresponding to the $(k-1, d, \Delta)$-mers (a, b) where $a \in p$ and $b \in p'$. While in the case of the exact distance and perfect coverage, red points define a collection of line segments in the multirectangle, in the case of inexact distance these points *fall into* a band of width Δ around these (unknown) red line segments. Thus, red points in each multirectangle should be somehow transformed into the red line segments, a difficult task. Below, we introduce the notion of $(k-1, d)$-tuple, which enables us to draw all possible positions of the red line segments, and later, using the red points in the multirectangle to classify these segments into correct/incorrect red line segments.

Given the de Bruijn graph DB, we define a $(k-1, d)$-*tuple* as a pair of $(k-1)$-mers $(a|b)$ such that there exists a path of length d between vertex a and vertex

b in DB. Obviously, every $(k-1,d)$-mer in *Genome* corresponds to a $(k-1,d)$-tuple in DB (but not vice versa).

Given a multirectangle formed by a pair of paths (p,p') and a collection of red points within it, we generate all possible [9] $(k-1,d)$-tuples $(a|b)$ such that $a \in p$ and $b \in p'$. These $(k-1,d)$-tuples define 45 degree line segments within this multirectangle. Those $(k-1,d)$-tuples that are also (k,d)-mers, form correct red line segments, while tuples that are not (k,d)-mers, form incorrect line segments. However, such a classification is unknown and we attempt to infer the correct/incorrect red line segments by the red points (corresponding to the $(k-1,d,\Delta$-mers) in the multirectangle.

Intuitively, correct line segments usually lie close to the "center" of red Δ-clouds, while the incorrect ones have few red points surrounding them. However, correctly classifying these segment into correct/incorrect segments still remains a difficult problem[10], since in the case of closely located red line segments, it is difficult to rule out which of them is correct (or whether they both are correct) and often forces us to combine such segments into a *a cluster* (see Fig. 3b) within an assumption that at least one of red line segments in the cluster represents a correct red segment. Below we describe the rectangle graph approach in the case when we deal with clusters of red segments.

In this case, we still represent each rectangle R as a single edge $edge(R)$ but use *multiple* labels for its starting and ending vertices (in the past we labeled these vertices by a single label). Specifically, we label its starting (ending) vertex by a multiset of all starting (ending) points of red segments. The *multilabeled rectangle graph* is defined as follows:

- Form a directed edge $edge(R)$ for each rectangle R. Starting (ending) vertices of $edge(R)$ are labeled by a *set* of labels of all starting (ending) points of the red segment within this rectangle.
- The rectangle graph is formed by gluing vertices in G if their sets of labels overlap.

Given a multirectangle R, SPAdes+ identifies T *clusters* of line segments that are supported by $ReadPairs$ using a variation of the approach from [3]. It further generates T rectangles (each rectangle with a single cluster of red segments as in Fig. 3b) and applies the multilabeled rectangle graph to assemble the resulted rectangles.

Missing Rectangles. We now consider the case when some rectangles are missing and ask whether the missing rectangles can be somehow reconstructed to complete the puzzle. Fig. 2d illustrates the case of 3 missing rectangles (R5, R6, and R7) resulting in a "gap" in the rectangle graph between vertices $(GTT|GAG)$ and $(ACG|GGT)$ in Fig. 2e. These *dead-ends* vertices (i.e., vertices with indegree or outdegree zero) provide a clue that some rectangles are missing and, as we show below, often allow one to recover the missing rectangles.

[9] If no such $(k-1,d)$-tuple exists, we remove the multirectangle.
[10] A similar problem was addressed in [3,12].

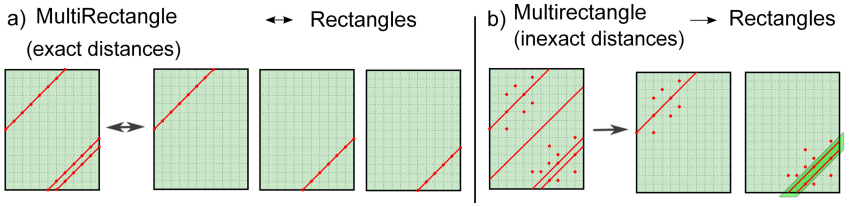

Fig. 3. Rectangles and red line segments. a) Multirectangle: an equivalent representation of multiple rectangles that are formed by the same non-branching paths but have different positions of the red line segments. b) A multirectangle is transformed into 2 rectangles (one of them represents a cluster of two closely positioned red line segments). Note that one segment (the longest) in the multirectangle was classified as incorrect (there is no red point around this segment) and further being removed.

Consider two points with integer coordinates $(x, x+d)$ and $(x+t, x+d+t)$ in the grid located at the intersection of the red line with the sides of the rectangles. We refer to labels of these points as $(a|b)$ and $(a'|b')$, correspondingly. For example, points (7,11) and (13,17) in Fig 2d correspond to paired $(k-1)$-mers $(GTT|GAG)$ and $(ACG|GGT)$. Given *Genome*, we define $Rectangles((x,y) \to (x',y'))$ as the set of all rectangles crossed by the segment of the red line between points $(x, x+d)$ and $(x+t, x+d+t)$. For example, $Rectangles((7,11) \to (13,17)) = \{R5, R6, R7\}$.

Fig. 2c presents an idealized case when *Genome* as well as the points (7,11) and (13,17) (that contain vertices from some missing rectangles) are known. In reality, this information is not available in genome assembly projects. However, one knows the de Bruijn graph and can guess the labels of the points (7,11) and (13,17) (as labels of the dead-ends vertices in the rectangle graph in Fig. 2e). This raises the question whether the missing rectangles $Rectangles((7,11) \to (13,17))$ can be inferred from the paired $(k-1)$-mers $(GTT|GAG)$ and $(ACG|GGT)$ (that represent labels of points (7,11) and (13,17)) *without* knowing the coordinates of these points. Given paired $(k-1)$-mers $(a|b)$ and $(a'|b')$, below we define the set of rectangles $Rectangles((a|b) \to (a'|b'))$ that often approximates $Rectangles((x,y) \to (x',y'))$ well.

Given an integer t, paired $(k-1)$-mers $(a|b)$ and $(a'|b')$ are called *t-tied* if there exist instances of a, b, a', b' located, respectively, at positions $x, y, x+t, y+t$ in *Genome*. Labels of every two red points in the grid represent t-tied paired $(k-1)$-mers. Below we relax the definition of t-tied paired $(k-1)$-mers for the case when *Genome* is unknown and only the de Bruijn graph of *Genome* is given.

Given an integer t and a de Bruijn graph DB, paired $(k-1)$-tuples $(a|b)$ and $(a'|b')$ are called *t-linked* if there exists a path $p = p_0 \dots p_t$ of length t between a and a' and a path $q = q_0 \dots q_t$ of the same length between b and b' in the de Bruijn graph DB. Obviously, every t-tied paired k-mers is also t-linked, but not vice versa. Paths p and q define $t+1$ paired $(k-1)$-tuples $(p_i|q_i)$ that may potentially belong to the red line (since the notion of "t-linked" is a relaxation of the notion of "t-tied"). We define $Rectangles_{p,q}((a|b) \to (a'|b'))$ as the set of all rectangles that contain at least one point $(p_i|q_i)$ (for $0 < i < t$). We will often

refer to $Rectangles_{p,q}((a|b) \rightarrow (a'|b'))$ as simply $Rectangles((a|b) \rightarrow (a'|b'))$ when it does not cause a confusion.

For example, $(GTT|GAG)$ and $(ACG|GGT)$ are 6-linked since there exists a path p (q) of length 6 from GTT to ACG (from GAG to GGT) in the de Bruijn graph in Fig. 2a. The vertices of the path $p(q)$ are located on 2 (3) non-branching paths in the de Bruijn graph thus contributing to $2 \times 3 = 6$ rectangles. Only 3 of these 6 rectangles ($R5, R6$, and $R7$) contain red points implying that $Rectangles((GTT|GAG) \rightarrow (ACG|GGT)) = \{R5, R6, R7\}$.

This example illustrates that one can close the gap between the dead-ends vertices $(a|b)$ and $(a'|b')$ in the rectangle graph by simply finding t-linked dead-ends in the rectangle graph (for small values of t), generating the set of missing rectangles $Rectangles((a|b) \rightarrow (a'|b'))$, and adding these missing rectangles to the pool of previously generated rectangles. Finally, one can construct the rectangle graph from the resulting enlarged set of rectangles.

4 Results

Assembly Datasets. To evaluate rectangle graph algorithm for genome assembly, we assembled two paired-end datasets from [5]. All these datasets are Illumina short reads with 100bp read length, 600× coverage. The first dataset is

Table 1. Comparison of assemblies for single-cell (ECOLI-SC) and standard (ECOLI-MC) datasets

Assembler	# contigs	NG50 (bp)	Largest (bp)	Total (bp)	Covered (%)	Misassemblies	Mismatches (per 100 kbp)	Complete genes
Single-cell *E. coli* (ECOLI-SC)								
EULER-SR	1344	26662	126616	4369634	87.8	21	11.0	3457
SOAPdenovo	1240	18468	87533	4237595	82.5	13	99.5	3059
Velvet	**428**	22648	132865	3533351	75.8	2	**1.9**	3117
Velvet-SC	872	19791	121367	4589603	93.8	2	**1.9**	3654
E+V-SC	501	32051	132865	4570583	93.8	2	6.7	3809
SPAdes-single	1164	42492	166117	4781576	**96.1**	1	6.2	3888
SPAdes	1024	49623	177944	4790509	**96.1**	1	5.2	3911
SPAdes+	509	**56842**	**209690**	4550761	95.5	**0**	3.6	**3975**
Normal multicell sample of *E. coli* (ECOLI-MC)								
EULER-SR	295	**110153**	221409	4598020	99.5	10	5.2	4232
SOAPdenovo	192	62512	172567	4529677	97.7	1	26.1	4141
Velvet	198	78602	196677	4570131	**99.9**	4	1.2	4223
Velvet-SC	350	52522	166115	4571760	**99.9**	**0**	1.3	4165
E+V-SC	339	54856	166115	4571406	**99.9**	**0**	2.9	4172
SPAdes-single	445	59666	166117	4578486	**99.9**	**0**	**0.7**	4246
SPAdes	195	86590	**222950**	4608505	**99.9**	2	3.7	4268
SPAdes+	192	91893	221829	4593658	**99.9**	2	3.5	**4274**

The best assembler by each criteria is indicated in bold. EULER-SR 2.0.1, Velvet 0.7.60, Velvet-SC, and E+V-SC were run with vertex size 55. SOAPdenovo 1.0.4 was run with vertex size 27–31. SPAdes-single refers to SPAdes without repeat resolution, (without using read-pairs information) for comparison with E+V-SC, which does not use read-pairs information. SPAdes, SPAdes-single and SPAdes-rectangle iterated over edge sizes $k = 22, 34, 56$.

the multiple cell *E.coli* dataset with average insert size 215 bp, and denoted as ECOLI-MC. The second dataset is single cell *E.coli* dataset with average insert size 266 bp.

Benchmarking. We compare our SPAdes+ algorithm with EULER-SR [4], SOAPdenovo [9], Velvet [13], Velvet-SC [5], E+V-SC [5] and SPAdes [3]. See Table 1. Our rectangle graph algorithm outperforms other assemblers in most metrics. Improvement is more significant for single cell dataset. On ECOLI-SC dataset, SPAdes+ produces contigs with higher N50 (56,842 bp vs 49,623 by SPAdes), with higher largest contig (209,690 bp vs 177,944 by SPAdes) and no misassemblies, also captures 64 additional *E. coli* genes (3975 vs 3911 by SPAdes).

For both *E. coli* datasets, the rectangle graph module works for less than 10 seconds and using less than 100 MB RAM given (1) the de Bruijn graph has been already constructed and (2) mapping information of all paired-end reads to the de Bruijn graph has been calculated (using other modules in [3].

5 Conclusion

In this paper, we modeled the problem of genome assembly using read-pairs as a simple jigsaw puzzle and reintroduced the notion of rectangle graph [3] in a more intuitive way. We further addressed algorithmic challenges that arise in the application of rectangle graphs that have not been addressed in [3]. We demonstrated that by addressing these algorithmic challenges, the quality of the assembly significantly improves for both single cell and multicell bacterial datasets.

Acknowledgements. Authors would like to thank all members of SPAdes team for productive collaboration. We are especially indebted to Dmitry Antipov, Paul Medvedev, Glenn Tesler and Anton Korobeynikov for their insightful comments.

References

1. Aardenne-Ehrenfest, T., Bruijn, N.G.: Circuits and trees in oriented linear graphs. Classic papers in combinatorics, 149–163 (1987)
2. Abrham, J., Kotzig, A.: Transformations of euler tours. Annals of Discrete Mathematics 8, 65–69 (1980)
3. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S., Prjibelski, A.D., et al.: Spades: A new genome assembly algorithm and its applications to single-cell sequencing. Journal of Computational Biology 19(5), 455–477 (2012)
4. Chaisson, M.J., Pevzner, P.A.: Short read fragment assembly of bacterial genomes. Genome Research 18(2), 324 (2008)
5. Chitsaz, H., Yee-Greenbaum, J.L., Tesler, G., et al.: Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. Nat. Biotechnol. 29(10), 915–921 (2011)

6. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23, 195–208 (2007)
7. Idury, R.M., Waterman, M.S.: A new algorithm for DNA sequence assembly. Journal of Computational Biology 2(2), 291–306 (1995)
8. Kampel, M., Sablatnig, R.: 3d puzzling of archeological fragments. In: Proc. of 9th Computer Vision Winter Workshop, vol. 2. Slovenian Pattern Recognition Society (2004)
9. Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., et al.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20(2), 265 (2010)
10. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. Journal of Computational Biology, 1625–1634 (2011)
11. Pevzner, P.A., Tang, H.: Fragment assembly with double-barreled data. Bioinformatics 17(suppl. 1), S225 (2001)
12. Pham, S.K., Antipov, D., Sirotkin, A., Tesler, G., Pevzner, P.A., Alekseyev, M.A.: Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in Genome Assembly. In: Chor, B. (ed.) RECOMB 2012. LNCS, vol. 7262, pp. 200–212. Springer, Heidelberg (2012)
13. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Research 18(5), 821 (2008)