Supplementary Material for
BILCO: An Efficient Algorithm for Joint Alignment of Time Series

Xuelong Mi[1], **Mengfan Wang**[1], **Alex Bo-Yuan Chen**[2], **Jing-Xuan Lim**[2],
Yizhi Wang[1], **Misha Ahrens**[2], **Guoqiang Yu**[1]
[1]Dept. of Electrical and Computer Engineering, Virginia Tech
[2]Howard Hughes Medical Institute, Janelia Research Campus
[1]{mixl18,mengfanw,yzwang,yug}@vt.edu
[2]{chena,limj2,ahrensm}@janelia.hhmi.org

Contents

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

S1 Proofs of lemmas and theorems

In this section, we show the proofs for lemmas and theorem in the main body of the paper. In addition, some new lemmas and theorems are given for a better understanding of our method.

S1.1 Proof of Lemma 1

Definition: We define the edges in graphical time warping (GTW) subgraph (Fig.1(c)(d)) as "forward edges", which are dual to edges in dynamic time warping (DTW) graph (Fig.1(b)). Their reverse edges are with infinite capacities due to the GTW structure.

Lemma 1: If two nodes are in the same component, then there is at least one path linking them.

Proof: By definition, a component is segmented by two adjacent cuts in one GTW subgraph. The nodes inside a component are connected by forward edges. For two nodes v and w, there must be one node u bridging v and w, so that (1) u could reach both v and w, or (2) both v and w could reach u, or (3) one of v or w could reach u, and u could reach another node. Since the component is one subset of the GTW subgraph, the capacities of all reverse edges are infinite. For any pair of nodes (v', w'), if v' could reach node w' through path consists of positive-direction edges, v' could also be reached by w' through reverse path. Thus, in any case above, v could reach w through u. ■

S1.2 Proof of Lemma 2

We claim the following two statements before proving Lemma 2. They are related to Fig.3 in the main body.

Statement 1: To solve the max-flow/min-cut problem, the flows on E_{cross} can be converted to extra edges linking to the source (entering flow) or linking to the sink (outgoing flow) whose capacity is the same as the amount of flow on E_{cross} (Fig.3(a)(b)).

Proof: This statement can be proved by the fact that all extra edges can be saturated when achieving max-flow. Due to the assumption of our algorithms, the graph conversion is only done for active components before pushing out. Thus, the components still have more excess to push out even with outgoing flows on E_{cross}. By converting the outgoing flows to edges linking to sink with the capacities of the same amount, all such edges will be saturated to achieve the max flow. Regarding entering flows on E_{cross}, they can be seen as excess received from neighboring GTW subgraphs. Thus, by converting them to edges linking to the source, the edges can also be saturated and accumulate within the component. The accumulation can be seen as excess flows from the source outside the component to the source within the component. Therefore, both extra edges linking to the source or linking to the sink can be saturated when achieving the max-flow and the flows on E_{cross} are converted. ■

Statement 2: If node v has an extra edge with capacity a linking to the sink t in GTW subgraph, then such an edge can be incorporated by adding weight a to all edges above v in the same column of DTW graph. Similarly, an extra edge on v linking to the source can be incorporated into the edges below v in the same column of DTW graph (Fig.3(b)-(f)). Before and after such incorporation for one node v, the corresponding cut values/path costs are the same.

Proof: Assume $G = \{V, E\}$ is one single GTW subgraph and $G' = \{V, E \cup (v, t)\}$ is one graph with extra edge on v linking to the sink t with capacity a. Let P be one arbitrary warping path on the DTW graph (dual graph of G) and its cost be c_P. Its corresponding cut in G will segment V into two parts V_S and V_T, and has the same cost $c_P = \sum_{(v,w)|v\in S, w\in T} c(v, w)$ due to the duality. Assume c'_P is the cost of same-location cut in G'. If P is higher than v, then $c'_P = c_P + a$ since v is classified to the V_S in G' and c'_P will count one more edge $c(v, t)$ (Fig.3(c)). If P is lower than v, then $c'_P = c_P$ since u is classified to the V_T in G' and no impact on the cost. Thus, all paths above the node v will increase cost a if v links to t with capacity a. By adding weight a on all the edges (right or inclined edges) above v in its column on DTW graph (Fig.3(d)(e)), the dual graph of modified DTW graph (Fig.3(e)(f)) is equivalent to G'. Similarly, all paths below the node v will increase cost a if v links to s with capacity a. To guarantee the equivalency, the edges below v should add weight a on DTW graph. Then, the cost of any warping path in the modified DTW graph is the same as its corresponding cut in G'. ■

Lemma 2: The corresponding cut values/path costs before and after graph conversion are the same.

Proof: By statement 1, all the flows on E_{cross} can be converted to the edges linking to the source or sink. Then by statement 2, the extra edge on any node can be incorporated without changing the values of cuts in the same location. Thus, by incorporating all extra edges, the corresponding cut values/path costs before and after graph conversion are the same. ■

S1.3 Proof of Lemma 3

Lemma 3: The amount of maximum possible excess on node v is the min-cut value in residual graph by taking v as sink.

Proof: The amount of maximum excess of one node v can carry is the max-flow from s to v in the residual subgraph after pushing flow to sink within the GTW subgraph. By duality between max-flow and min-cut, the value is also equal to the min-cut value between s and v. ■.

In residual graph, the cost of min-cut above v (between v and t) is 0, the cost of min-cut below v (between s and v) is the excess. Regarding the original graph, the amount of maximum possible excess is the difference between the min-cut value below v and the min-cut value above v, since s locates on the bottom-right side and t locates on the top-left side of GTW subgraph. Note that any cut in GTW subgraph is corresponding to one path in its dual DTW graph. Therefore, working on the modified DTW graph (Fig.3(e)), the excess value can be obtained by calculating the difference between the shortest path below v and the shortest path above v (the true shortest one), as shown in Algo.4.

S1.4 Proof of Lemma 4

Lemma 4: All nodes in the same component have the same label.

Proof: By Lemma 1, for any pair v and w in the same component, there are always path $(v, v_1, v_2, ..., w)$ and path $(w, w_1, w_2, ..., v)$ with no cross edge. By Equation (2a) in the main body, $d(v) \leq d(v_1) \leq ... \leq d(w)$ and $d(w) \leq d(w_1) \leq ... \leq d(v) \Rightarrow d(v) = d(w)$. Thus, all the nodes in the same component are with the same label. ■

S1.5 Proof of Lemma 5

Lemma 5: The new labeling function is consistent with generic push-relabel labeling function if treating each component as one unit.

Proof: If the validity of the new labeling function holds, then the validity of the generic push-relabel labeling function also holds by treating each component as one unit. Assume $v \in R_1, w \in R_2$, the residual capacity of (v, w) is positive, then $R_1 \rightarrow R_2$. If components R_1 and R_2 are in neighboring subgraphs, then by Equation (2b) $d(v) \leq d(w) + 1$. That means, $d(R_1) \leq d(R_2) + 1$. If components R_1 and R_2 are in the same subgraph, then by Equation (2a) $d(v) \leq d(w)$. That means, $d(R_1) \leq d(R_2) \leq d(R_2) + 1$. Thus, maintaining the new labeling function can also keep the validity of the generic push-relabeling function if treating each component as one unit. ■

Then, our strategy can be seen as one alternative push-relabel approach that uses the component as the operational unit. All the statements proved in [5] hold on the component level, including the correctness. Our method can achieve max-flow in polynomial component operation.

S1.6 Proof of Lemma 6

Lemma 6: Assume the min-cut segments the nodes V into source side V_S and sink side V_T, then replacing the nodes in V_S or V_T by source or sink does not impact the min-cut.

Proof: Let G^f be the residual graph after pushing the max-flow. By definition, there is no edge link from V_S to V_T in G^f. Replacing the nodes of V_S in G^f by source, there is still no flow that can be sent to sink. Thus, there is no change on the min-cut. Similarly, replacing the nodes in V_T by sink also does not impact the min-cut. ■.

S1.7 Proof of Lemma 7

Lemma 7: Assume V_T represents the nodes of sink side segmented by the real min-cut of original GTW graph, V_{T_1} denotes the sink side segmented by the obtained min-cut of the graph under replacing arbitrary node by the source s, then $V_{T_1} \subseteq V_T$.

Proof: Assume original graph is $G = \{V, E\}$ and $G_1 = \{V_1, E_1\}$ is the graph after replacing some nodes in V by the source s. Let's denote their corresponding residual graphs after achieving max-flow as G^f and G_1^f, respectively. By definition of the residual graph, all the nodes in V_{T_1} can reach the sink along some path consisting of residual edges in G_1^f. While in the original graph G, the flow entering V_{T_1} could not be larger than the one in G_1. Thus, the residual edges connecting V_{T_1} in G^f will not be saturated. The nodes in V_{T_1} could still reach the sink in G^f and will be divided into the sink side. Therefore, $V_{T_1} \subseteq V_T$. ■.

S1.8 Proof of Theorem 1

Theorem 1: The obtained min-cut in bidirectional strategy is the min-cut of the original graph.

Proof: Through Lemma 7, $V_{T_1} \subseteq V_T$ after pushing excess in one direction. Then, replacing nodes in V_{T_1} by the sink t, the real cut will not change according to Lemma 6. Thus, the obtained min-cut in bidirectional strategy is the min-cut of the original graph. ■.

S1.9 Additional lemmas and proofs

In addition to the listed lemmas and theorems above, there are also some useful lemmas and theorems that play important roles in excess pushing with linear component operations (ELCO) and need to be proved. Lemma 8 explains why the part below the found min-cut is active and another part is inactive. Lemma 9 shows how the property of the labeling function helps us implement the relabel step faster. And Theorem 2 guarantees that the labeling function maintains valid in the whole ELCO process.

Lemma 8: Assume the min-cut (the one closest to the source if there are multiple equivalent min-cuts) segments the graph into two parts V_S and V_T, where $s \in V_S$ and $t \in V_T$. Then any node in V_S could hold excess.

Proof: By the definition of min-cut, all nodes in V_S segmented by the chosen min-cut could be reached by source s through some path in the residual graph. Pushing excess through such a path, any node in V_S could hold excess. ■

That's the reason why we choose the lowest min-cut in Algo.5. Then, the new segmented component below the min-cut is active, while the segmented component above the min-cut is inactive.

Lemma 9: If component R_1 is higher than component R_2 or in the left side of R_2 in the same GTW subgraph, then $d(R_1) \leq d(R_2)$.

Proof: By the property of subgraph, the capacities of reverse inner edges are infinite. If component R_1 is higher than R_2 or in the left side of R_2 in the same subgraph, then there must be one path that only consists of reverse edges from the node $v \in R_1$ to node $w \in R_2$, since the reverse edges in one GTW subgraph are with infinite capacities. By Equation (2a), $d(R_1) \leq d(R_2)$.

Lemma 9 could help us implement the relabel step in Algo.2 quickly.

Theorem 2: ELCO algorithm maintains the invariant that d is a valid labeling.

Proof: To check the validity of labeling function, we need to check Equations (2a) and (2b) on E_{within} and E_{cross} for all steps:

"Drain" and "Discharge" operations (Algo.3 and Algo.4): "Drain" and "Discharge" operations will not change the label of nodes within the original component. Finding the new cut within subgraph means the constraint Equation (2a) is loosened. Thus, the labeling is maintained and Equation (2a) holds. For "Drain" operation, no label is changed and no new cross edges are generated, thus Equation (2b) holds. For "Discharge" operation, pushing excess through cross edge (v, w) may loosen the constraint in pushed edge (saturate such edge) but result in more strict limitation in reverse direction (generate reverse edge due to pushing). Assume the pushed edge is (v, w), and the generated reverse

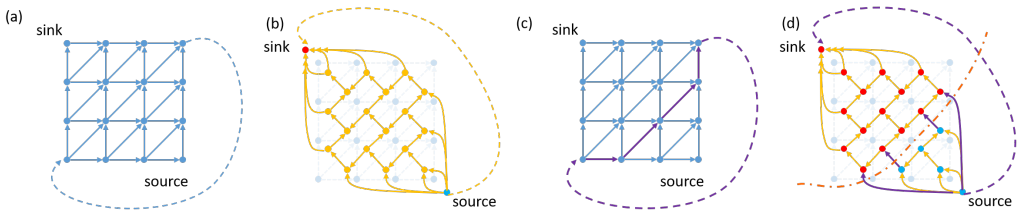

Figure S1: (a) An example of DTW graph with the auxiliary edge, which is represented by the dashed line. (b) Dual graph of (a). The dashed line is the dual edge of the auxiliary edge in (a). (c) An example cycle (purple) for graph (a). (d) The dual graph of (c), where the purple cut is dual to the cycle in (c).

edge is (w, v) where $d(v) = d(w) + 1$. Since $d(w) \leq d(v) + 1 = d(w) + 2$, Equation (2b) always holds. Thus, both "Drain" and "Discharge" maintains the validity of the labeling function.

Split operation (Algo.5): Regarding splitting components with positive labels, no label is changed and no new edges are generated. Thus, both Equations (2a) and (2b) hold. While Algo.5 will change the label when splitting 0-label component. For 0-label component, assume the part above new cut is R_1, the part below new cut is R_2, and $d(R_1) = 0$, $d(R_2)$ is changed to 1. We will focus on R_2 since only these labels are changed. For the edge pairs of (R_1, R_2), $d(R_1) \leq d(R_2)$, Equation (2a) holds. Assume there is component R_3 that R_2 and R_3 are linked. If R_2 and R_3 are in the same GTW subgraph, and $R_2 \rightarrow R_3$, then by algorithm (the previous split operation), $d(R_3) \geq 1$, and $d(R_2) = 1 \leq d(R_3)$, Equation (2a) holds. If R_2 and R_3 are in the same GTW subgraph and $R_3 \rightarrow R_2$, Equation (2a) holds before splitting means $d(R_3) = 0$. Then $d(R_3) = 0 \leq d(R_2) = 1$, Equation (2a) holds. Thus, the validity of E_{within} is kept. If R_2 and R_3 are not in the same GTW subgraph and $R_2 \rightarrow R_3$, then $d(R_3) \geq 0 \Rightarrow d(R_2) = 1 \leq d(R_3) + 1$, Equation (2b) always holds. If R_2 and R_3 are not in the same GTW subgraph and $R_3 \rightarrow R_2$, Equation (2b) holds before splitting means $d(R_3) \leq d(R_2) + 1$. $d(R_2)$ increases after splitting, Equation (2b) must hold. Thus the split operation maintains the validity.

Relabel: as shown in Algo.2, $d(R) = d_{minWithin}$ satisfies Equation (2a) and $d(R) = d_{minCross} + 1$ satisfies Equation (2b). The relabeled value $\min(d_{minWithin}, d_{minCross} + 1)$ satisfies both, thus validity is maintained. ■

Thus, the validity of the new labeling function is always maintained for ELCO method.

S1.10 Duality between DTW graph and GTW subgraph

Lemma 10: For any cut C in GTW subgraph, the corresponding dual edges can compose a warping function P in DTW graph so that $cost(C) = cost(P)$, and vice versa.

Proof: Linking an auxiliary edge from the top-right node to the bottom-left node (Fig. S1(a)), dual DTW graph can be constructed (Fig. S1(b)). Each node in GTW subgraph is a face in DTW graph, and each edge e' in GTW subgraph connects the face from the right side of e to the left side with the same weight in DTW graph. According to the dual graph theory [1], cycles are dual to cuts. For any cycle in DTW graph (Fig. S1(a)(c)), there is a corresponding cut in GTW subgraph (Fig. S1(b)(d)), and vice versa. Deleting the auxiliary edge from two graphs, it's easy to establish a one-to-one correspondence between the warping path P in DTW graph and the cut C in GTW subgraph. Since the edges composing C and P are dual, $cut(C) = cost(P)$. ■

S2 Algorithms

Before discussing the algorithms, we need to introduce some annotations first. For node v, $f_{in}(v)$ denotes the total entering flow of v on E_{cross} and $f_{out}(v)$ denotes the total outgoing flow of v on E_{cross}. $f(v, w)$ denotes the flow from v to w, $c_f(v, w)$ denotes residual capacity of (v, w) in GTW graph. And if we use edge (a, b), we denote the edge on the DTW graph, which is the dual graph of (converted) GTW subgraph. Both a and b represents one coordinate position like (x, y). $c(a, b)$ or $c'(a, b)$ denotes the cost of edge on the DTW graph before graph conversion or after graph conversion. Besides, regarding region R, we denote its upper bound and lower bound by $P_{upper}(R)$ and $P_{lower}(R)$.

Algorithm 1 Convert(R)

Let $c'(a,b) = c(a,b)$ for all edges $(a,b) \in$ corresponding DTW graph of R, $c(a,b)$ is its cost
for each column from left to right **do**
  sourceModify = 0 ▷ Save the accumulative value for source edge
  for each node v from top to bottom **do** ▷ Top-down modification
    if $f_{in}(v) > f_{out}(v)$ **then** ▷ Entering flow is larger
      sourceModify = sourceModify + $f_{in}(v) - f_{out}(v)$
      $c'(a,b) = c'(a,b)$ + sourceModify, where (a,b) is the most adjacent edge below v
    end if
  end for
  sinkModify = 0 ▷ Save the accumulative value for sink edge
  for each node v from bottom to top **do** ▷ Bottom-up modification
    if $f_{in}(v) < f_{out}(v)$ **then** ▷ Outgoing flow is larger
      sinkModify = sinkModify + $f_{out}(v) - f_{in}(v)$
      $c'(a,b) = c'(a,b)$+sinkModify, where (a,b) is the most adjacent edge above v
    end if
  end for
end for
Return w' as the modified edges for dual graph (DTW graph) of converted GTW subgraph.

Algorithm 2 DP(R, c)

(x_0, y_0) is the bottom-left point of R
Initialize dynamic matrix D, $D(x_0, y_0) = 0$
for each column of from left to right ($x \uparrow$) **do**
  $x-1$ and x are the x-coordinate boundary of this column
  for each row from bottom to top ($y \uparrow$) **do**
    $y-1$ and y are the x-coordinate boundary of this row
    Let $b = (x, y)$
    Let $a_x = (x-1, y), C_x = D(x-1, y) + c(a_x, b)$
    Let $a_{xy} = (x-1, y), C_{xy} = D(x-1, y-1) + c(a_{xy}, b)$
    Let $a_y = (x, y-1), C_y = D(x, y-1) + c(a_y, b)$
    $D(x, y) = \min(C_x, C_{xy}, C_y)$
  end for
end for
Return D

S2.1 Graph conversion

Algo. 1 incorporates flows on E_{cross} by two-pass modifications for each column. The top-down update incorporates all source edges since each entering flow on E_{cross} is similar to adding an edge linking to the source and would impact all edges below, as shown in Fig.3(a)-(e). Through top-down update, the entering flows can be incorporated by one pass. On contrary, the outgoing flows can be incorporated by one-pass bottom-up update. Thus, the graph conversion can be done in linear time complexity.

S2.2 DP on DTW graph

The dynamic matrix D can be calculated in linear time through DP, and each of its elements denotes the shortest distance between the bottom-left starting point and current position. Then, the shortest

Algorithm 3 Drain(R)

c' = Convert(R) ▷ Graph conversion
D = DP(R, c')
Find shortest path P from D by tracking back from the top-right end
Return min-cut, where min-cut is dual to P

Algorithm 4 Discharge(R)

c' = Convert(R) ▷ Graph conversion
Calculate D_{rev} matrix through DP on DTW graph with edges c'. (Reverse direction of Algo. 2)
(x_0, y_0) is the bottom-left point of R
$vMinCut = D_{rev}(x_0, y_0)$.
Initialize dynamic matrix D, $D(x_0, y_0) = 0$
for each row from bottom to top ($y \uparrow$) **do**
  $y - 1$ and y are the y-coordinate boundary of this row
  Let $a_y = (x0, y - 1)$, $b = (x_0, y)$
  $D(x_0, y) = D(x_0, y) + c'(a_y, b)$
end for
for each column from left to right ($x \uparrow$) **do**
  $x - 1$ and x are the x-coordinate boundary of this column
  Initialize vector $c_{below} = inf, c_{pre} = inf$ ▷ Initialize the min-cut value below each node
  for each node v from bottom to top **do** ▷ Calculate the min-cut value below each node
    (a, b) is the edge below v, $a = (x_a, y_a), b = (x_b.y_b)$
    $c_{curPath} = D(x_a, y_a) + D_{rev}(x_b.y_b) + c'(a, b)$ ▷ The cost of adjacent path below v
    $c_{pre} = \min(c_{pre}, c_{curPath})$
    $c_{below}(v) = c_{pre}$
  end for
  for each node v from top to bottom **do**
    Excess $e = c_{below}(v) - vMinCut$. ▷ Calculate excess on v
    initialize $send = 0$ ▷ Record the pushed out excess
    if $e > 0$ **then**
      Flow pushed out $e_p = 0$.
      for $\{w | v \to w, v \in G^n, w \notin G^n\}$ **do** ▷ same node but in neighbor GTW subgraphs
        if $d(w) = d(v) - 1$ **then**
          $e_h = \min(e, c_f(v, w))$
          $f_{in}(w) = f_{in}(w) + e_h$, $f_{out}(v) = f_{out}(v) + e_h$ ▷ Push from v to w
          $f(v, w) = f(v, w) + e_h$, $c_f(v, w) = c_f(v, w) - e_h$ ▷ Push from v to w
          $e_p = e_p + e_h$, $e = e - e_h$, $send(v) = e_p$ ▷ Update
        end if
      end for
    end if
    $vMinCut = vMinCut + e_p$ ▷ New outgoing flow, min-cut value increases
  end for
  initialize $send_{ac} = 0$ ▷ Accumulated modification in this column
  for each row from bottom to top ($y \uparrow$) **do** ▷ Update D one this column
    $y - 1$ and y are the y-coordinate boundary of this row
    $b = (x, y)$
    $a_x = (x - 1, y)$, v_x is the node adjacent below (a_x, b), $send_{ac} = send_{ac} + send(v_x)$
    $C_x = D(x - 1, y) + c'(a_x, b) + send_{ac}$
    $a_{xy} = (x - 1, y)$, v_{xy} is the node adjacent below (a_{xy}, b), $send_{ac} = send_{ac} + send(v_{xy})$
    $C_{xy} = D(x - 1, y - 1) + c'(a_{xy}, b) + send_{ac}$
    $a_y = (x, y - 1)$
    $C_y = D(x, y - 1) + c'(a_y, b)$
    $D(x, y) = \min(C_x, C_{xy}, C_y)$
  end for
end for
Find shortest path P from D by tracking back from the top-right end
Return min-cut, where min-cut is dual to P

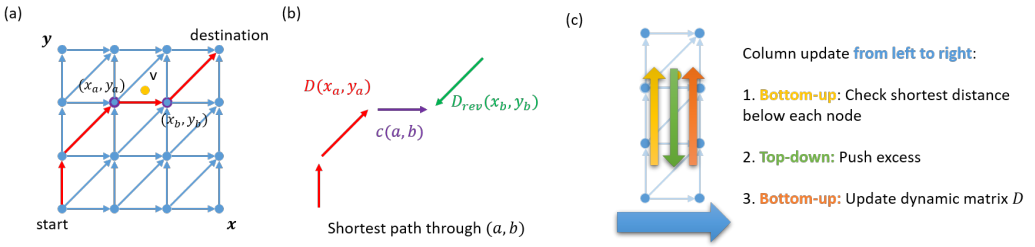

Figure S2: Details in "Discharge" operation. (a) Dual DTW graph and node v. The red path is the shortest path through edge (a, b). (b) The diagram shows how to calculate the shortest path cost through edge (a, b). (c) Three-pass updating in each column.

path can also be quickly found in $\Theta(\sqrt{N})$ by tracking back from the top-right end of the dynamic matrix D.

S2.3 "Drain" operation

"Drain" operation only consists of graph conversion and DP, as shown in Algo. 3. Since each one is linear-time complexity, "Drain" operation can be implemented in linear time.

S2.4 "Discharge" operation

The "Discharge" operation consists of four steps: (1) Incorporating flows on E_{cross} (Graph conversion), which can be done in linear time. (2) Calculating reverse dynamic matrix D_{rev} through DP in linear time, whose each of the elements represents the shortest distance from top-right to the current position. (3) Updating D from left to right. In each column (Fig.S2), there is one-pass bottom-up checking the min-cut value below each node, one-pass top-down pushing excess out to other GTW subgraphs, and another one-pass bottom-up updating dynamic matrix D in each column (considering the pushed flow). (4) Finding the shortest path from D and returning the corresponding new min-cut. The first three steps can be implemented in linear time, while the last step can be implemented with less complexity. Thus, "Discharge" operation can be implemented in linear time.

Specifically, "Discharge" operation contains more than one layer of DP. We will discuss why DP can implement the linear "Discharge" component operation in the following content.

As mentioned in Lemma 3, the maximum possible excess on node v is the min-cut value in the residual graph by taking v as the sink. Regarding the primal graph, the value is equal to the difference between the min-cut between v and s and the real min-cut between s and t. That is the difference between the shortest path distance below v and the true shortest path on the dual DTW graph. The shortest path crossing one edge (a, b) in the dual DTW graph can be calculated as $D(x_a, y_a) + D_rev(x_b, y_b) + c(a, b)$ (Fig.S2(b)), where D and D_{rev} are two dynamic matrices recording the shortest path from bottom-left and top-right respectively. Through bottom-up checking (Fig.S2(c)), the shortest path distance below one node v can be calculated, and then the excess can be obtained. That's corresponding to the first one-pass bottom-up checking in step (3), where the values of shortest path distance can be recursively used. With that result, the excess on each node can be calculated.

Excess can be pushed in any order, either top-down or bottom-up. Here we choose top-down order so that there is a large chance for excess to reach the position closer to the sink since the sink is in the top-left position of each GTW subgraph. This corresponds to the top-down pushing in step (3) (Fig.S2(c)).

Then, we can normally update the dynamic matrix D by DP from bottom to top (Fig.S2(c)). But remark that pushing excess out will further change the flow on E_{cross} and impacts the converted graph. That means, it will change the values of D_{rev} in the current column. Thus, in the outer layer of Algo.4, we push nodes from left to right, so that the values in D_{rev} we will use later will not be impacted. Both dynamic matrices D and D_{rev} will only be calculated once, and thus "Discharge" operation can be implemented in linear-time complexity.

Algorithm 5 Split(R, cut)

Split R into K disjoint components $\{R_k|k = 1, 2, ..., K\}$ according to cut, $P_{upper}(R)$ and $P_{lower}(R)$
for all R_k higher than cut **do**
  Mark R_k inactive
  $P_{upper}(R_k) = P_{upper}(R), P_{lower}(R_k) = cut$
  $d(R_k) = d(R)$
end for
for all $R_k \prec cut$ **do**
  Mark R_k active
  $P_{upper}(R_k) = cut, P_{lower}(R_k) = P_{lower}(R)$
  if $d(R) = 0$ **then**
    $d(R_k) = 1$
  else
    $d(R_k) = d(R)$
  end if
end for

S2.5 "Split" operation

Algorithm 6 Gap-relabel

Check the gap between the labels of components.
If not finding the gap, turn. Otherwise, the gap is d_gap.
for all $R_i|d(R_i) > d_gap$ **do**
  $d(R_i) = |V|$
  Set (R_i) unselectable.
end for

It's worth noting that the cut we selected for each "Drain" and "Discharge" operation is the lowest cut if there are multiple equivalent min-cuts. Then, given such a min-cut, we can ensure the segmented components above cut are inactive, and segmented components below cut are active (Fig.2(c)). This can be easily proved by calculating the excess of nodes as shown in step (3) of Algo.4. Besides, for 0-label components, the labels of new segmented active components are set to 1. That's to ensure the property that all 0-label components still have the ability to drain excess, and it won't break the validity of the labeling function as proved in Theorem 2.

S2.6 Gap-relabel heuristic

Algo.S2.5 is the same as the gap-relabel heuristic in generic push-relabel but on the component level, setting the components with the label higher than the gap unselectable. According to the labeling function and the rule of pushing, the component with label d can only push excess to the component with label $d - 1$. That means no matter how to push the excess, the components with the label higher than the gap cannot send flow to the sink.

S3 Redundancy in push-relabel method

For push-relabel-based methods, if some excess cannot reach the sink, it will still be pushed back and forth many rounds until the nodes/components carrying them are relabeled high enough, as Fig.S3 shows. Then, they can be absorbed by the source and the push-relabel method could terminate. These redundant steps can not make a real influence on the finial result. Even using the gap-relabel heuristic, the process of identifying the gap is still redundant, especially when a large number of nodes are connected and the excess on them cannot reach the sink. The excess has to be conveyed through each connected node until the minimum label of those nodes is larger than the real gap.

S4 Illustrative example

Here we show one simple example of joint alignment problem with 2 time series and 4 time points as shown in Fig. S4. Since our algorithm majorly works on the dual graph of GTW subgraph, i.e., DTW graph, here we show the cut, component, pushing operations on the DTW graphs. We also have an animation attached in the supplementary to demonstrate this example and explain how the cut is calculated.

S5 Implementation

S5.1 Implementation of peer methods

We use the following implementations for our comparison, and all of them can be found in our supplemental material or GitHub website.

- **Incremental breadth-first search (IBFS)**: We use the C++ implementation designed by Goldberg in [4] and the MATLAB wrapper is provided by Anton https://github.com/aosokin/graphCutMex_IBFS.
- **Hochbaum's pseudoflow (HPF)**: We use the C implementation described in [6] https://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html. MATLAB wrapper is also provided in the package.
- **Boykov-kolmogorov (BK) max-flow**: BK method was developed by Boykov and Kolmogorov [2] and the C++ code with MATLAB wrapper was provided by Anton https://github.com/aosokin/graphCutDynamicMex_BoykovKolmogorov.
- **Highest-label push-relabel (HIPR)**: This algorithm was implemented in C language by Andrew Goldberg [5, 3](v3.5, http://www.avglab.com/andrew/soft.html). The implementation has two stages while only the first stage for finding the min-cut is used in our comparison. The MATLAB wrapper was implemented by our team and also attached in the supplemental material.

S5.2 Implementation of BIdirectional Pushing with Linear Component Operations (BILCO) method

We implement our algorithm in C++ with a MATLAB wrapper. Due to the requirement of anonymity, here we only attach our code in the supplemental material. In the camera-ready version, we will publish our code on GitHub.

S5.3 Memory usage estimation

We have shown the memory comparison between BILCO and peer methods in Table 1. In the following, we will explain how we obtain those values. It is worth noting that each element of E we use includes both forward edge and reverse edge. But for other max-flow methods, each edge structure only store one directed edge.

From the implementation of BILCO and peer methods mentioned in previous subsections, the memory usage of each node and edge can be obtained as shown in Table S1. Each input edge contains two

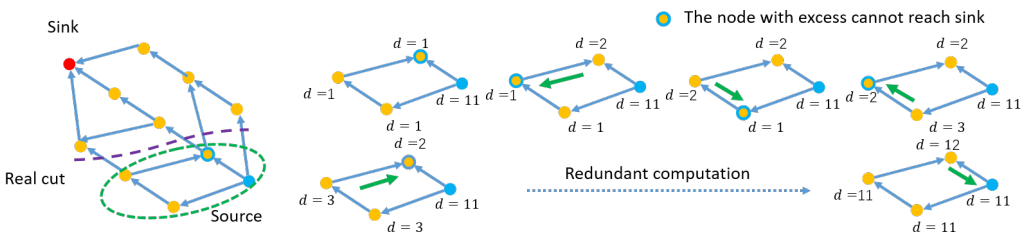

Figure S3: Diagram of the common issues in push-relabel-based methods.

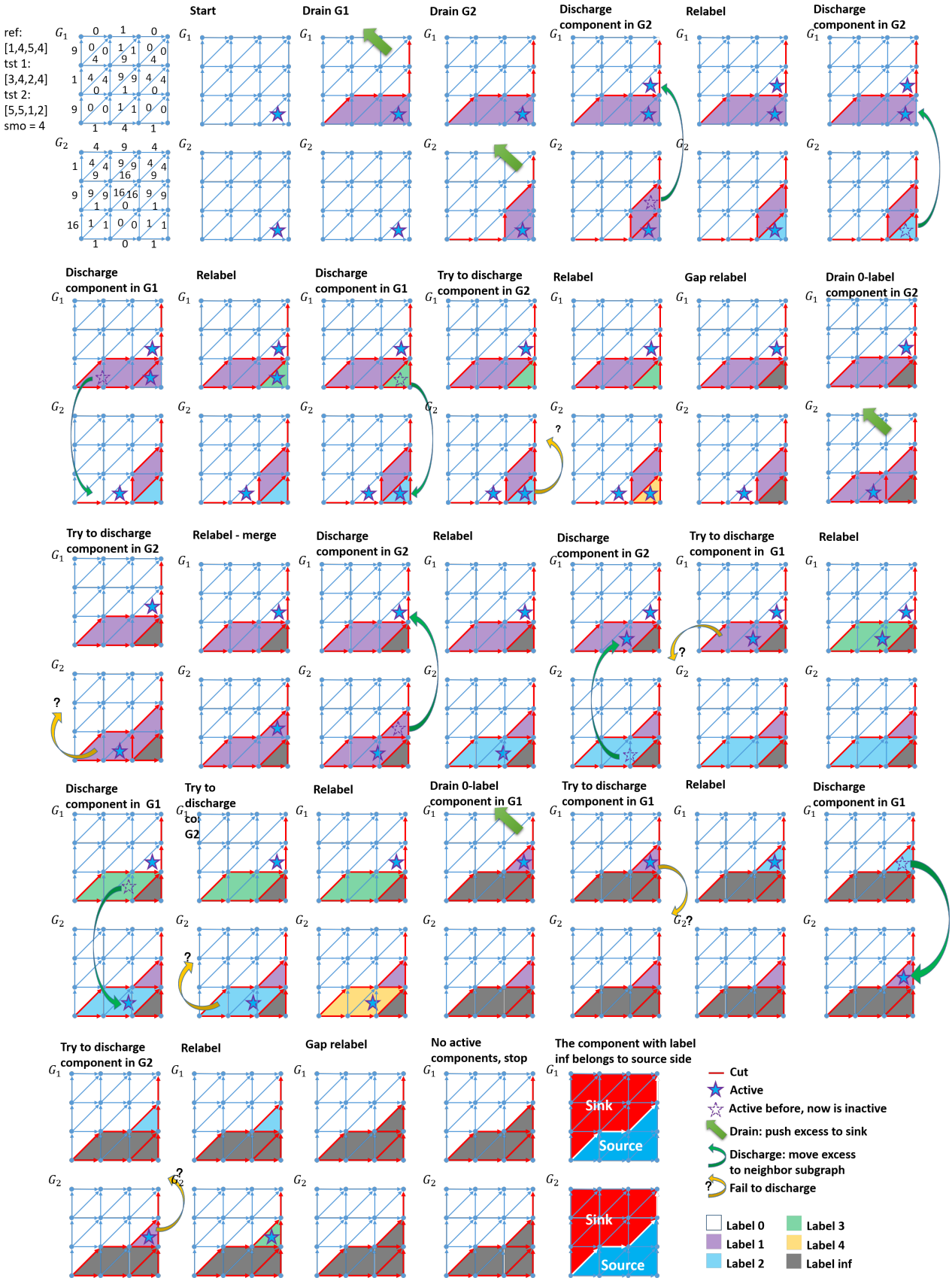

Figure S4: An illustrative example.

Table S1: Memory usage (bytes) of node and edge among IBFS, HPF, BK, and HIPR.

	IBFS	HPF	BK	HIPR
Node	48	88	48	48
Edge	32	32	32	24

directed edges, the memory of IBFS and BK can be calculated as the same $48|V| + 64|E|$. HPF and HIPR are more complex. There are extra memory costs such as saving "buckets" in HIPR, or saving "roots". Especially, for HIPR, it stores every directed edge individually and further creates a new edge as the corresponding reverse edge. Each input edge we use will correspond to four edge structures in HIPR. Counting all extra cost, HPF and HIPR cost the memory $156|V| + 96|E|$ and $64|V| + 112|E|$, respectively.

For BILCO algorithm, it takes the structure of GTW graph into account. For each node, it uses 8 bytes (the length of pointer) to record their corresponding component address, 4 bytes (float type) to store f_{in} and f_{out} (save in the same vector. Positive and negative values represent entering flow and outgoing flow), 2 bytes to record the cost of one point in the warping function (like the distance between $x[k_1]$ and $y[k_2]$. Two nodes are in the same block of DTW graph thus each element of cost (float type, 4 bytes) is corresponding to two nodes). Besides, we also record the flows on E_{cross}, each edge will cost 4 bytes. However, due to the structure of GTW graph, $|E| = |E_{within}| + |E_{cross}| \approx 1.5|V| + |E_{cross}|$ (2 nodes are corresponding to one block in the dual DTW graph, averagely 3 edges for one block). Since the smoothness in Equation (1) is the same, the capacities of all edges are the same and there is no need to record the capacity for each edge. And the bucket is set dynamic, compared to the $|V|$ or $|E|$, the memory usage is neglectable. Counting all the necessary memory, BILCO costs $8|V| + 4|E|$ bytes.

S6 Experiments

In this section, we state the empirical hyperparameter setting in Section S6.1, then explain the details of each experiment (synthetic data, real data on three distinct categories) in Section S6.2 - S6.5. More experimental results are given in Section S6.6.

$cost(P_n)$ in Equation (1) may use the different distance measures. Here we use function $g(x[k_1], y[k_2])$ to represent the cost of aligning the k_{1th} time point of time series x and the k_{2th} time point of time series y together in the following settings. That means, $cost(P_n) = \sum_{(k_1,k_2)\in P} g(x[k_1], y[k_2])$. In different experiments, the distance metric is different.

S6.1 Hyperparameter setting

The hyperparameter κ in Equation (1) adjusts the balance between the pairwise profile similarity of single time series pair and the warping function distance between two neighbor pairs. Since in all the experiments we've done preprocessing step to normalize the data, we set κ to 0.2 or 1, an empirical setting for good alignment performance.

S6.2 Setting for synthetic data

We generate the synthetic data mimicking the real propagation in [12]. The bell-shaped signal first appears in the center of a 2D grid, then propagates to the boundaries with the same speed in each direction. The intensity of the signal keeps the same during the propagation, while Gaussian noise is added with 10 dB signal-to-noise (SNR) ratio. The smoothness of warping functions was encouraged in 4-connected neighbor pixels, of which the GTW graph is just like the one in Fig. 1(d). To use the empirical setting of κ, the curves of all the pixels are normalized by dividing the standard deviation of the noise. And all the curves are aligned with the same referenced curve, which is the signal template without noise. The distance metric is set by squared error, i.e., $g(x[k_1], y[k_2]) = (x[k_1] - y[k_2])^2$.

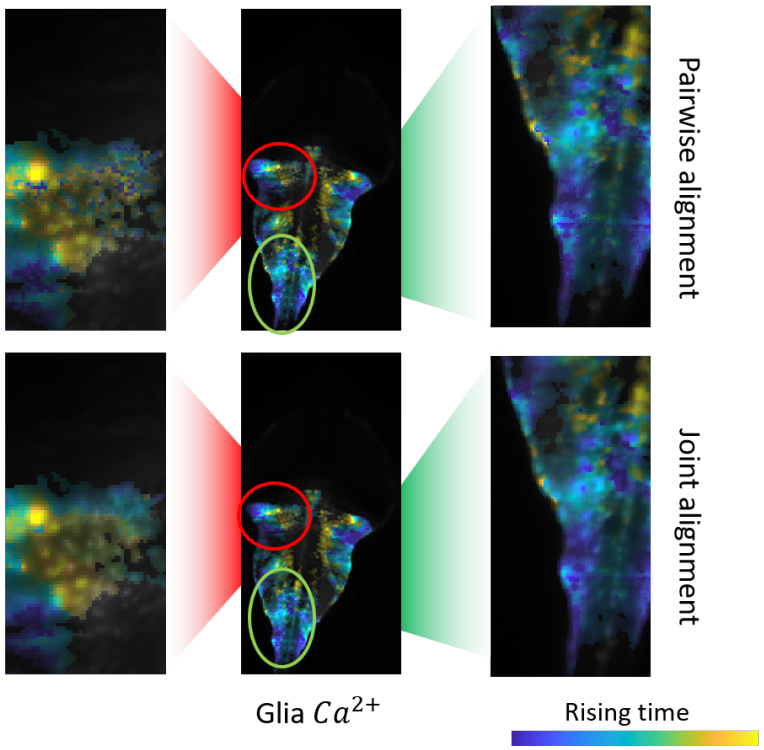

Figure S5: The obtained signal propagation of one example dataset. Different color shows the different rising time, while the intensity of color shows the strength of the signal. If there is nearly no signal, it will show a gray background.

Figure S6: Results of extracting depth information. The top row, second row, and the last row show the left view of binocular stereo, pairwise alignment result, and joint alignment result.

S6.3 Setting for application to calculating signal propagation and results

Data acquisition: The "Glia Ca^{2+}" data is one crop data collected in the experiment following the similar setting of [7]. There are 24411 time series pairs with curve length 150. The downsample "Glia $Ca^{2+}*$" is the spatial downsampling data of the former with resizing parameter 0.25. That is, "1570" time series pairs with the curve length. "5_rat_astro_ATP" data is one crop of public data in [10], with 8243 time series pairs and 70 time points. "Exvivo Ca^{2+}" data is one crop public data in [11] that needs to align 6794 time series pairs and 70 time points.

Preprocessing: This application is aimed to calculate the signal propagation, where background intensity is not considered. Thus, for each dataset, we estimated the background image first (moving average) and then removed it from the original data. To apply the empirical κ setting, we normalized each pixel curve by its noise standard deviation. The processed reference curves and test curves can be found in supplemental materials.

Alignment setting: We used the same referenced curve for all pixels in the same dataset to calculate the signal propagation. That is, the curves of all pixels were aligned to the same template. The template is calculated by averaging all the curves to align first and then smoothed later. Due to the strong signal in these datasets, a small $\kappa = 0.2$ is enough to constrain the distance of warping functions and give a good signal propagation result. We used the squared error as the distance metric in this application $(g(x[k_1], y[k_2]) = (x[k_1] - y[k_2])^2)$.

Alignment results: Fig.S5 shows the obtained signal propagation pattern of one example dataset. Though the high resolution makes it hard to find the difference in the large scope, in a small scope the artifacts in pairwise alignment can be found while joint alignment shows a good result. For a low SNR scenario as Fig.1(e), where $\kappa = 0.2$, the necessity of joint alignment is obvious. From the result of joint alignment, we can definitely find two different signals and detect the boundary between them, while from pairwise alignment, it's hard to discover the same thing. And under such case, if do smoothing on the result of pairwise alignment, the boundary will be blurred and two signals may be considered as one signal. That demonstrates the necessity of joint alignment.

S6.4 Setting for extracting depth information and results

Data acquisition: All the binocular stereo data in this application are collected from [9], where two images of the same scene are recorded and processed with image rectification. The depth information is inversely proportional to the binocular disparity, which is the misalignment of the same object only in the horizontal direction. Thus, through alignment technique, the depth information can be derived. The original data is 1080×1920, while the downsampled data is 324×576. They can be found in supplemental materials.

Preprocessing: Since our BILCO is designed for general use of joint alignment and not purposely designed for dealing with RGB data, the input data is converted to gray data in preprocessing step.

Alignment setting: We align each row of two images of the same scene, where the smoothness is encouraged between adjacent rows according to the smooth surface assumption. The absolute error is adopted as the distance cost in this application $(g(x[k_1], y[k_2]) = |x[k_1] - y[k_2])|$. Besides, we set a window size of $1/5$ sequence length to avoid unnecessary computation.

Alignment results: Fig.S6 shows the depth information extracted through pairwise alignment and joint alignment. Though joint alignment also contains some artifacts, it is much better than pairwise alignment and can provide enough depth information for the major targets. The performance of such an application is very hopeful to be improved if we take three channels of RGB data into account.

S6.5 Setting for signature identification and results

Data acquisition: The signatures are obtained from [8], where only the first 20 people's signatures are used in our experiment. There are 25 genuine signatures and 25 forgeries for each person, thus totally we have compared 1000 signatures. Each signature has five dimension feature sequences: x-coordinate, y-coordinate, pressure, pen azimuth, and pen elevation. All these features are collected under a sampling rate $100Hz$. The lengths of sequences of the same signature are the same, while different signatures may have different lengths, even if they belong to the same person. Due to

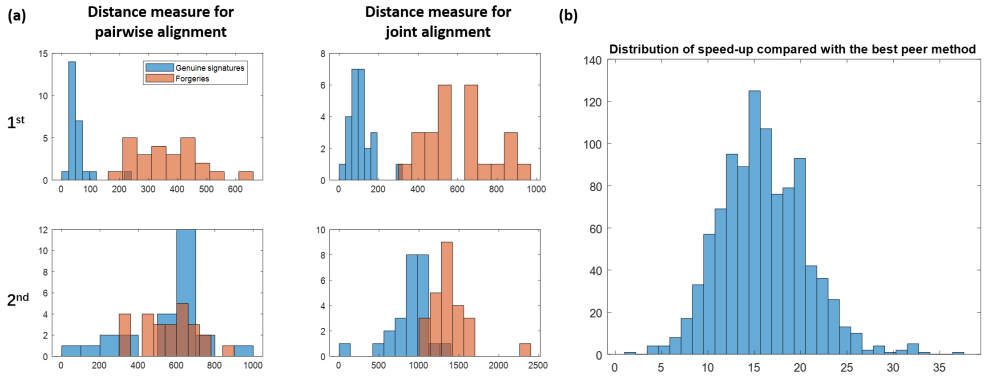

Figure S7: - (a) Distance measure distributions for genuine signatures and forgeries under pairwise alignment and joint alignment from first two persons. (b) Distribution of BILCO speed-up compared with the best peer method in 1000 signature identifications.

the license limitation, we won't provide the signature data in our supplemental material. It can be required from http://atvs.ii.uam.es/atvs/mcyt100s.html.

Preprocessing: To balance the impact of different sequences, we normalize the data by following two steps: (1) Subtract the mean value from each sequence. (2) Divide the standard deviation of each sequence. For example, the feature sequence x is processed by $x' = \frac{x - mean(x)}{std(x)}$.

Alignment setting: For each person, we used one genuine signature as the reference pattern to match other signatures from the same person. The same features are aligned, while smoothness is imposed between different features. We adopt the squared error as the distance cost in this application $(g(x[k_1], y[k_2]) = (x[k_1] - y[k_2])^2)$. The hyperparameter κ is set to be an empirical setting 0.2.

Alignment results: Fig.S7(a) compares distance measure distributions under pairwise alignment and joint alignment for first two persons' signatures. It's obvious to find that the genuine signatures and forgeries are more separable under joint alignment, this demonstrates the necessity of utilizing dependency information. Fig.S7(b) also shows how many folds speed up BILCO can present compared with the **best** peer method in 1000 comparisons, where BILCO provides an averagely 15-fold speed-up.

S6.6 Complementary results

We've done more experiments on the application of calculating signal propagation and extracting depth information, the efficiency and results are listed in Table.S2, Fig.S8 and Fig.S9.

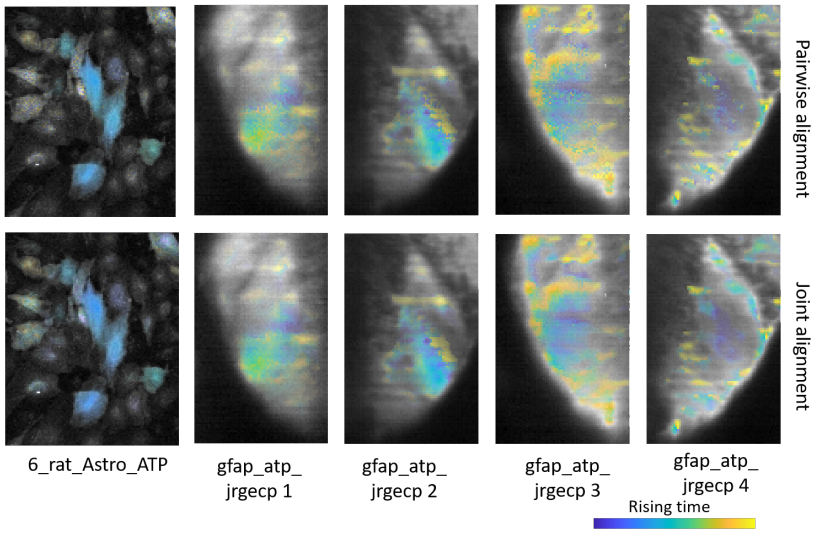

Figure S8: The obtained signal propagation of complementary experiments. Different color shows the different rising time, while the intensity of color shows the strength of the signal. If there is nearly no signal, it will show the gray background.

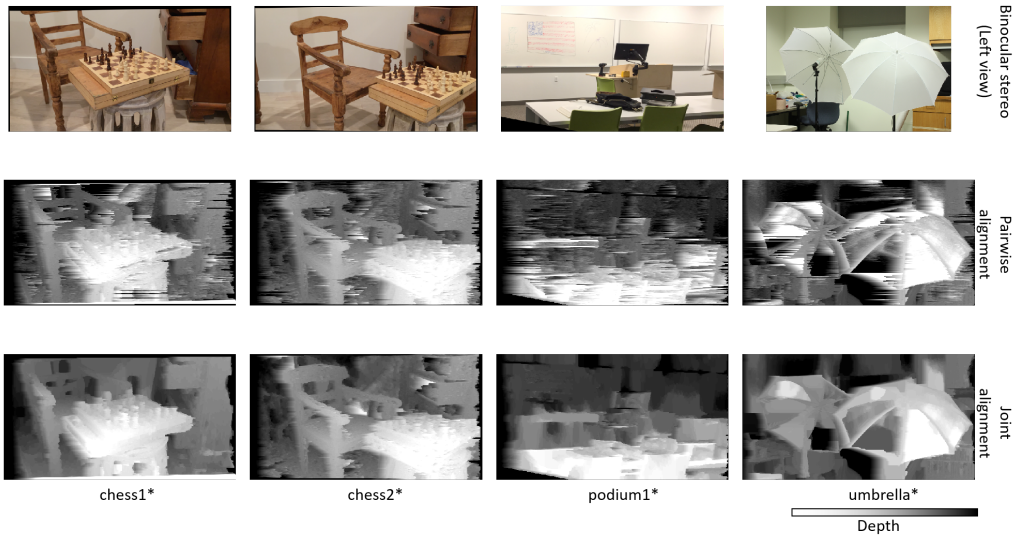

Figure S9: Results of extracting depth information for complementary experiments. The top, the second, and the last row shows the left view of binocular stereo, pairwise alignment result, and joint alignment result.

References

[1] RK Ahujia, Thomas L Magnanti, and James B Orlin. Network flows: Theory, algorithms and applications. *New Jersey: Rentice-Hall*, 1993.

[2] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.

[3] Boris V Cherkassky and Andrew V Goldberg. On implementing push-relabel method for the maximum flow problem. In *International Conference on Integer Programming and Combinato-*

rial Optimization, pages 157–171. Springer, 1995.

[4] Andrew V Goldberg, Sagi Hed, Haim Kaplan, Robert E Tarjan, and Renato F Werneck. Maximum flows by incremental breadth-first search. In *European Symposium on Algorithms*, pages 457–468. Springer, 2011.

[5] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[6] Dorit S Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations research*, 56(4):992–1009, 2008.

[7] Yu Mu, Davis V Bennett, Mikail Rubinov, Sujatha Narayan, Chao-Tsung Yang, Masashi Tanimoto, Brett D Mensh, Loren L Looger, and Misha B Ahrens. Glia accumulate evidence that actions are futile and suppress unsuccessful behavior. *Cell*, 178(1):27–43, 2019.

[8] Javier Ortega-Garcia, J Fierrez-Aguilar, D Simon, J Gonzalez, Marcos Faundez-Zanuy, V Espinosa, A Satue, I Hernaez, J-J Igarza, C Vivaracho, et al. Mcyt baseline corpus: a bimodal biometric database. *IEE Proceedings-Vision, Image and Signal Processing*, 150(6):395–401, 2003.

[9] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *German conference on pattern recognition*, pages 31–42. Springer, 2014.

[10] Yinxue Wang, Guilai Shi, David J Miller, Yizhi Wang, Congchao Wang, Gerard Broussard, Yue Wang, Lin Tian, and Guoqiang Yu. Automated functional analysis of astrocytes from chronic time-lapse calcium imaging data. *Frontiers in neuroinformatics*, 11:48, 2017.

[11] Yizhi Wang, Nicole V DelRosso, Trisha V Vaidyanathan, Michelle K Cahill, Michael E Reitman, Silvia Pittolo, Xuelong Mi, Guoqiang Yu, and Kira E Poskanzer. Accurate quantification of astrocyte and neurotransmitter fluorescence dynamics for single-cell and population-level physiology. *Nature neuroscience*, 22(11):1936–1944, 2019.

[12] Yizhi Wang, David J Miller, Kira Poskanzer, Yue Wang, Lin Tian, and Guoqiang Yu. Graphical time warping for joint alignment of multiple curves. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3655–3663, 2016.

Table S2: Efficiency Comparison of different methods on real data.

Problem			BILCO	IBFS	HPF	BK	HIPR
Name	|V|	|E|	Time Memory	Time/(Time of BILCO) Memory/(Memory of BILCO)			
Calculate signal propagation in imaging data (the third one is from [10], others are generated by us)							
6_rat_astro_ATP[10]	3.1×10^7	1.0×10^8	**19s** **0.7GB**	×11.2 ×11.3	×10.8 ×19.9	×435.3 ×11.3	×43.8 ×17.4
gfap_atp_jrgecp 1	1.8×10^8	6.2×10^8	**200s** **4.5GB**	×9.5 ×10.5	×13.2 ×18.2	**>**12hours ×10.5	×54.7 ×18.4
gfap_atp_jrgecp 2	9.9×10^7	3.4×10^8	**43s** **2.2GB**	×18.9 ×11.9	×26.5 ×20.7	×405.1 ×11.9	×168.8 ×16.1
gfap_atp_jrgecp 3	2.2×10^8	7.5×10^8	**238s** **5.2GB**	×8.8 ×11.1	×15.5 ×17.8	**>**12hours ×11.1	Out of Memory
gfap_atp_jrgecp 4	1.8×10^8	6.4×10^8	**112s** **4.3GB**	×17.4 ×11.5	×26.0 ×20.2	**>**12hours ×11.5	×81.3 ×20.1
Extract depth information in binocular stereo (data from [9], no window size is limited)							
chess1*	2.1×10^8	5.4×10^8	**120s** **3.9GB**	×15.3 ×10.6	×142.3 ×20.3	>12h ×10.6	×229.6 ×17.2
chess2*	2.1×10^8	5.4×10^8	**34s** **3.8GB**	×29.5 ×10.9	×133.3 ×21.4	×217.8 ×10.9	×413.6 ×19.6
podium1*	2.1×10^8	5.4×10^8	**166s** **3.8GB**	×6.4 ×11.0	×25.5 × 21.4	×157.7 ×11.0	×99.7 × 19.4
umbrella*	9.5×10^8	2.4×10^9	**1135s** **16.5GB**	Out of Memory	Out of Memory	Out of Memory	Out of Memory