Fast Combinatorial Vector Field Topology

Jan Reininghaus, Christian Löwen, and Ingrid Hotz

Abstract—This paper introduces a novel approximation algorithm for the fundamental graph problem of combinatorial vector field topology (CVT). CVT is a combinatorial approach based on a sound theoretical basis given by Forman's work on a discrete Morse theory for dynamical systems. A computational framework for this mathematical model of vector field topology has been developed recently. The applicability of this framework is however severely limited by the quadratic complexity of its main computational kernel. In this work we present an approximation algorithm for CVT with a significantly lower complexity. This new algorithm reduces the runtime by several orders of magnitude, and maintains the main advantages of CVT over the continuous approach. Due to the simplicity of our algorithm it can be easily parallelized to improve the runtime further.

Index Terms—9.VI.IX.II Flow Visualization, 7.II.II.I Graph algorithms

1 Introduction

Topological data analysis has been proven successful for the visualization of vector fields. The topology of a vector field is derived from the homotopy classes of its streamlines. Standard algorithms for the extraction of this topological skeleton involve many numerical challenges: finding all zeros, integrating streamlines and streamsurfaces, the intersection of those, and the extraction of periodic orbits. While there are stable numerical algorithms to do this [1], the overall resulting framework has many computational parameters that may strongly influence the result. Furthermore, the topological skeleton is usually rather complex for real world vector fields. One is therefore interested in a meaningful simplification of the skeleton [2], [3], [4], [5], [6].

A computational framework for a combinatorial vector field topology (CVT) which addresses these challenges was developed recently [7]. This approach has three main advantages compared to the continuous approach:

1) *Persistency*: The natural output of this framework is a hierarchy of topological skeletons. The importance of a critical point in this hierarchy is determined by a value closely related to the concept of persistence [8], [9]. This allows the user to discriminate between stable and unstable features of the vector field.
2) *Consistency*: The resulting topological skeletons are always consistent with the topology of the underlying domain. While this property may seem rather academic it significantly increases the robustness of the algorithm: a single critical point cannot be missed or misclassified, as this would affect the consistency of the result. In a sense, topological consistency serves as an error correcting code.
3) *Simplicity*: There are no computational parameters. This enables a fully automatic analysis of a series of vector fields.

The main weakness of this framework is its computational complexity of $O(n^2 \log n)$, where n denotes the number of nodes in the data set. Even rather small datasets with 60k nodes take about 30 minutes to compute. The quadratic scaling in the runtime therefore severely limits the applicability to real world data sets - an application to large 3D data sets seems also unfeasible using this algorithm.

We therefore propose a new algorithm that can replace the computational kernel of [7]. This approximation algorithm has a lower empirical complexity of $O(n^{3/2} \log n)$ and maintains the three main advantages mentioned above. It reduces the runtime by several orders of magnitude for our data sets, while it produces visually the same results as the exact algorithm and preserves a certain monotonic behavior of the exact solution. Almost all computational time of our algorithm is spent solving shortest path problems with negative weights. As this particular graph problem is easily parallelizable, we have implemented it using CUDA, which reduces the runtime even further.

While the proposed algorithm might in principal be applicable to vector field data of any dimension, we restrict its description and the analysis of its results to triangulated 2D manifolds.

2 Related Work

Vector field topology was introduced to the visualization community by Helman and

• *J. Reininghaus, C. Löwen and I. Hotz are with Zuse Institute Berlin. E-mail: reininghaus@zib.de, loewen@zib.de and hotz@zib.de*

Hesselink [10]. They defined the concept of a topological skeleton consisting of critical points and connecting separatrices to segment the field into regions of topologically equivalent streamline behavior. Algorithms to extract periodic orbits, completing this topological structure, were proposed in [11], [12], [6]. A good introduction to the concepts and algorithms of vector field topology is given in [1].

To improve the applicability of vector field topology, a variety of extensions like topology tracking, extraction of boundary topology, or extensions to 3D have been developed. For a rather complete overview of these methods we refer to the survey paper [13].

As the topological skeleton of real world data sets is usually rather complex, a lot of work has been done towards simplification of topological skeletons of vector fields, see [2], [3], [4], [5], [6].

Scalar field topology developed almost independently from vector field topology. The main application areas in visualization include segmentation, transfer function design, and ridge extraction. Due to their robustness and stability, combinatorial extraction algorithms have been especially successful in this context [14], [15], [16], [17]. To reduce the often very complex topological structure that is generated by these algorithms, a controlled simplification is introduced based on the mathematically well-founded concept of persistence in [8], [9]. Due to the simplicity and clarity of this simplification strategy, it has been widely adopted. Most of the above-mentioned extraction algorithms make use of Forman's work [18] on a discrete scalar field topology for cell complexes. Rather than choosing a suitable class of continuous functions, a single number is assigned to each cell of the complex and all further steps are combinatorial.

Motivated by the success of the concept of persistence and the simple and robust extraction algorithms [19] based on Forman's work [20], a computational framework for a combinatorial vector field topology introduced by Forman [21] was recently introduced in [7].

In this computational framework, the problem of extraction and simplification is reduced to a graph-theoretic problem, see Section 3.2. The fundamental graph problem (5) that needs to be solved is a generalized version of the maximum weighted bipartite matching problem [22]. In [7] this problem was solved exactly using the Hungarian method, see Section 4.1, with a computational complexity of $O(n^2 \log n)$. As for large data sizes this is prohibitively expensive, one is interested in fast approximation algorithms. There is a lot of literature on the approximation of the maximum weighted matching problem, see [23] for an overview. It is, however, unclear how one would efficiently extend these algorithms to compute the sequence of maximal weight matchings (5). Also, the graphs described in Section 3.2 have a very specific structure in their connectedness and weights that can be exploited by a custom algorithm.

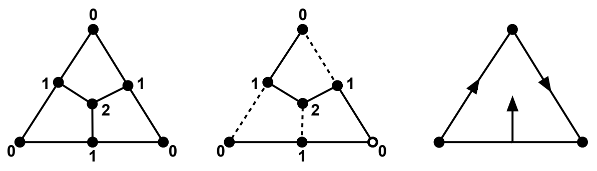

Fig. 1. Basic definitions. Left: the simplicial graph of a single triangle. Middle: a combinatorial vector field (dashed) - covered nodes are drawn solid. Right: equivalent depiction using arrows.

3 Foundation

In this section we briefly introduce the main concepts of combinatorial vector field topology (CVT). For a more detailed description and motivation of these ideas see [7]. We first present the mathematical model of CVT provided by Forman's work in Section 3.1. The computational framework containing a detailed description of the fundamental graph problem of CVT is introduced in Section 3.2.

3.1 Mathematical Model

For simplicity we restrict ourselves to triangulated 2D manifolds while the mathematical theory for CVT is defined in a far more general setting [21].

Given a triangulation of a manifold, we first define its simplicial graph $G = (S, L)$. The nodes S of the graph consist of the vertices, edges, and triangles of the triangulation and each node α^p is labeled with the dimension p of the geometric simplex it represents (see Figure 1, left). We denote the total number of nodes $|S|$ in this graph with n. We refer to the graph edges as links to avoid confusion with the edges of the triangulation. The links L of the graph encode the neighborhood relation of the triangulation: if the simplex represented by node α^p is in the boundary of the simplex represented by the node β^{p+1} then $\{\alpha^p, \beta^{p+1}\}$ is a link in the graph. Note that only simplices whose dimension differs by one are linkable.

A matching of a graph is defined as a subset of links such that no two links are adjacent. Using these definitions, a *combinatorial vector field* V can be defined as a matching of a simplicial graph (see Figure 1, middle). An arrow representation of this combinatorial field as used in [21] is shown in Figure 1 right.

The nodes of the graph that are not covered by V are called critical points (see Figure 2, left). If α^p is a critical point of V, we say that the critical point

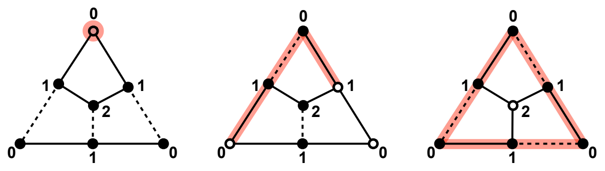

Fig. 2. Topological Features in CVT. Left: a critical point of index 0, i.e. a sink (red). Middle: a separatrix (red). Right: an attracting periodic orbit (red).

has index p. A critical point of index p is called sink $(p = 0)$, saddle $(p = 1)$, or source $(p = 2)$.

A combinatorial p-streamline is a path in the graph whose links alternate between V and the complement of V and the dimension of the nodes of the path alternates between p and $p + 1$. A p-streamline connecting two critical points is called a separatrix, see Figure 2, middle. If a p-streamline is closed, we call it either an attracting periodic orbit $(p = 0)$ (see Figure 2, right) or a repelling periodic orbit $(p = 1)$.

For a vector field defined on a 2D manifold K, the relation of the topological skeleton to the topology of the domain K can be easily formulated. Let c_p denote the number of critical points with index p, A_p the number of closed p-streamlines and b_p the p-th Betti number of the domain K. For a detailed introduction to the Betti numbers of a manifold we refer to [24]. Forman has proven in [21] that for any p there holds the strong Morse inequality

$$A_p + c_p - c_{p-1} + ... \pm c_0 \geq b_p - b_{p-1} + ... \pm b_0. \quad (1)$$

Due to the finite nature of the above definitions, the topological features (critical points, separatrices and periodic orbits) can be computed exactly in a combinatorial vector field. We therefore always get a topological skeleton that is consistent with the topology of the underlying manifold.

Note that in continuous vector field topology separatrices are sometimes defined differently - a streamline that connects a saddle with a periodic orbit is also called a separatrix. While this does not directly correspond to our definition of separatrices, we can also extract these lines by following the p-streamlines that emanate from a saddle.

3.2 Computational Framework

Given a combinatorial vector field on a 2D simplicial graph we can easily extract its complete topological skeleton, i.e. its critical points, separatrices and periodic orbits [25]. The main challenge of CVT lies therefore in the computation of a good combinatorial representative of the input vector field. For an overview of the computational framework that is described in detail below, we refer to Figure 3.

For simplicity, assume that we are given a continuous vector field f defined on a triangulated 2D manifold K. We first construct the simplicial graph $G = (S, L)$. To represent the continuous input data on this discrete entity, we compute link weights $\omega : L \to \mathbb{R}$. If $\ell = \{\alpha^p, \beta^{p+1}\} \in L$ is a matching edge it can be thought of as an arrow pointing from α^p to β^{p+1}. We therefore assign a large weight to ℓ if such an arrow reflects the flow behavior of f well. In this paper, we propose to measure the tangential flow of f along ℓ. Let $c(\cdot)$ denote the midpoint coordinates of a given simplex. Then the weight for a link is computed by integrating the vector field f in tangential direction $\tau = c(\beta^{p+1}) - c(\alpha^p)$ along ℓ, i.e.,

$$\omega(\ell) = \int_\ell f \cdot \frac{\tau}{\|\tau\|} \, ds. \quad (2)$$

If we are given a gradient vector field, then this definition corresponds to the difference of the scalar values given at the end points of the link. This is a standard definition of weights in scalar field topology [19].

A combinatorial representative V of the continuous input vector field f can now be computed by finding the heaviest matching of all matchings $\mathcal{M}$ of this link weighted-graph G

$$V = \arg \max_{M \in \mathcal{M}} \omega(M), \quad (3)$$

where $\omega(M)$ denotes the sum of the weights of the links in M. Note that the graph G is bipartite, i.e., there is a partition of its nodes $S = U \dot{\cup} W$ such that $L(U) = L(W) = \emptyset$, where $L(U)$ denotes the set of links whose nodes are contained in U. This greatly simplifies the algorithms needed to solve (3). A simple bipartition is given by $U = \{\alpha^p \in S : \ p \text{ is odd}\}$ and $W = \{\alpha^p \in S : \ p \text{ is even}\}$.

The size of the set of critical points $C(V)$ of a combinatorial vector field V is directly related to the number of links in its matching $|V|$ by $|C(V)| = |S| - 2\,|V|$. We can therefore compute a combinatorial vector field with a prescribed number of critical points by computing

$$V_k = \arg \max_{M \in \mathcal{M}, |M| = k} \omega(M), \quad (4)$$

i.e., the heaviest matching of a given size. Note that $C(V_{j+1}) \subset C(V_j)$ for any j due to the graph-theoretic structure of the matching problem (4), see [22]. Let $k_0 = |V|$ denote the size of the heaviest matching, and let $k_n = \max_{k \in \mathbb{N}} |V_k|$ denote the size of the heaviest maximum cardinality matching. From a topological point of view, V_{k_0} corresponds to the initial combinatorial vector field without any simplifications, while V_{k_n} corresponds to a completely simplified version of it.

The topological hierarchy $\mathcal{V}$ of a combinatorial vector field can now be defined as the sequence of

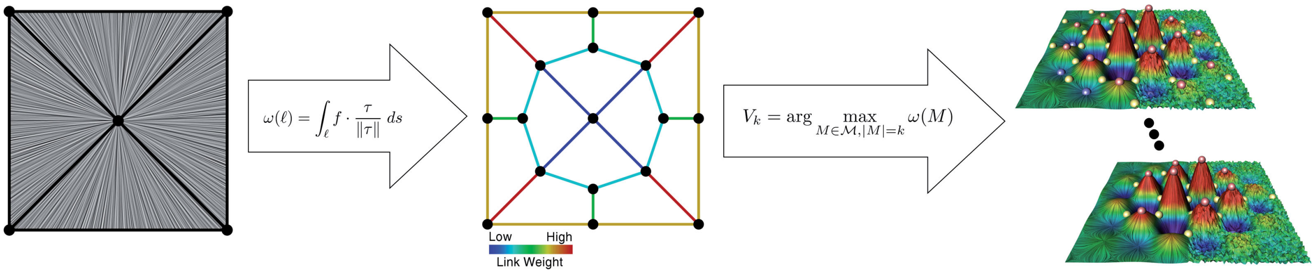
$\omega(\ell) = \int_\ell f \cdot \frac{\tau}{\|\tau\|}\, ds$
$V_k = \arg \max_{M \in \mathcal{M}, |M|=k} \omega(M)$
Low
High
Link Weight

Fig. 3. Computational pipeline. Given a triangulated vector field (left), we construct its corresponding simplicial graph (middle) and compute link weights using (2). We then compute the maximum weight matching sequence (4) of this link weighted graph resulting in a hierachy of topological skeletons (right) of the input data.

matchings

$$\mathcal{V} = (V_k)_{k=k_0,\dots,k_n} . \quad (5)$$

The main task of a computational framework for CVT is to compute the sequence (5). We therefore refer to it as the *fundamental graph problem* of CVT.

Note that the weight difference $\omega(V_j) - \omega(V_{j+1})$ gives us a measure for the importance of the simplification which is closely related to persistence [14]. For future reference, we call this difference of matching weights the *weight of cancellation* j. If we are dealing with gradient vector field data, then this weight difference corresponds to the difference of the scalar values of the critical points being canceled. We can therefore make use of (5) to remove topological noise (see Figure 6), or to reduce the topological representation of the data to its dominant structures (see Figure 7). This enables a multi-scale topological analysis of vector fields.

In [7] the fundamental graph problem of CVT was computed exactly using the Hungarian method [22]. While the Hungarian method is usually employed to compute the maximum weight matching (3), it naturally computes the whole sequence of maximum weighted matchings (5) we are interested in.

4 Algorithm

In this section we first briefly present the exact approach to the solution of the fundamental graph problem of CVT (5) employed in [7]. We then use this description of the exact method to motivate our approximation algorithm in Section 4.2. To enable a good reproducibility of our results we give a detailed description of our algorithm including pseudo code in Section 4.3. We conclude this section with some comments and details on the parallelization of this algorithm in Section 4.4.

Due to the structure of the considered graphs the number of links m is bounded by the number of nodes n, i.e. $O(n) = O(m)$. When discussing the complexity of the employed algorithms we will implicitly make use of this fact.

4.1 Exact Method

The following presentation of the Hungarian method [26] closely follows [22]. The basic idea is to start with V_0 and then to iteratively compute the sequence (4). In each iteration the augmenting path of maximum weight is computed. An augmenting path of a matching V_j is a path in the graph whose start and end nodes are not covered by the matching and whose links alternate between V_j and its complement. The weight of an augmenting path p is defined as the alternating sum $\sum_{\ell=0}^{|p|-1} (-1)^\ell \omega(p_\ell)$ of the weights of its links p_ℓ. Given an augmenting path p of maximum weight we can augment the matching V_j to get V_{j+1} by computing the symmetric difference $\triangle$ of V_j and p. This will result in V_{j+1}, see [22] for a proof. For an illustration of a single iteration of this method we refer to Figure 4.

The maximum weight augmenting path of a bipartite graph $G = (S, L)$ with a matching M can be found by solving a shortest path problem on a derived graph as follows. Let U and W denote a bipartition of the graph $G = (S, L)$, i.e. $S = U \dot\cup W$ and $L(U) = L(W) = \emptyset$. Further, let $D_M = (S, \vec{L})$ denote the directed graph with link weights $\vec{\omega}$ obtained from G and M by orienting each link $\ell \in M$ from W to U with weight $\vec{\omega}(\ell) := \omega(\ell)$, and orienting each link $\ell \in L \setminus M$ from U to W with weight $\vec{\omega}(\ell) := -\omega(\ell)$. See Lines 8 - 15 of Algorithm 4 for pseudo code of this construction. The augmenting path of maximum weight in M is now given as the shortest path in D_M

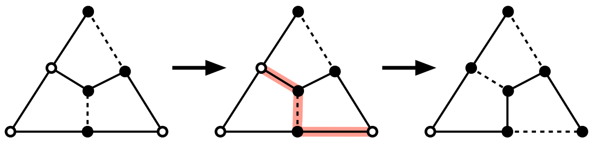

Fig. 4. Illustration of a single iteration of the exact method. Given the heaviest matching with two links (left) its heaviest augmenting path (middle) is computed. We augment the matching along this path to get the heaviest matching with three links (right).

from the set of nodes $U_M := U \setminus S(M)$ to the set of nodes $W_M := W \setminus S(M)$, where $S(M)$ denotes the nodes of S that are covered by M.

Note that the sets U_M and W_M are precisely the critical points of the combinatorial vector field given by the matching M. From a topological point of view one could therefore say that the Hungarian method iteratively simplifies the vector field by canceling pairs of critical points.

One can easily see that D_M contains no circle C with negative weight because of the maximality property of the computed matchings (4). One can therefore employ the Bellman-Ford algorithm [22] to find the shortest path. This leads to a total runtime of $O(n^3)$ for the fundamental graph problem of CVT (5). One can also employ Dijkstra's algorithm (even though the graph contains negative weights) by constructing a graph with nonnegative weights with the same shortest path from U_M to W_M. To do this efficiently, one updates a node potential in each iteration, which leads to an overall runtime of $O(n^2 \log n)$ for (5). This is the method that was employed in [7].

Note that the weight of the maximum weight augmenting path decreases as we iteratively compute (5), see [22]. In the following approximation algorithm we make sure that this monotonic behavior is preserved.

4.2 Approximation Method

The exact algorithm described above has a computational complexity of $O(n^2 \log n)$, as it needs to solve n shortest path problems with positive weights. To reduce this rather large runtime we propose to approximate the exact solution by augmenting the matching along all shortest paths computed when we solve the shortest path problem.

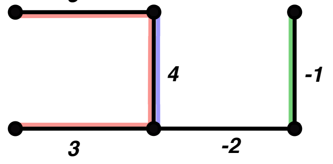

Fig. 5. Comparison of algorithms for (5). A simple edge weighted graph illustrates the differences between the presented algorithms. The order of augmenting paths taken by the exact algorithm is (blue, red, green) with corresponding weights (4, 2, -1). The approximate algorithm without the corrector phase takes (blue, green, red) with corresponding weights (4, -1, 2). The full Predictor-Corrector algorithm works as follows: (blue, green, undo green, red, green) resulting in the monotone sequence of augmenting path weights (4, 2, -1).

As demonstrated in Section 5.4 this leads to a significantly reduced overall runtime. For more details on the algorithmic realization of this idea we refer to Section 4.3 and the pseudo code given in Algorithm 2. We call this step of our algorithm the *Predictor* phase.

If we continued in this fashion we would generate an approximation of (5) without the monotonic behavior mentioned at the end of Section 4.1. In practice this may result in a bad approximation of the exact solution (see Figure 6, bottom-left). We therefore maintain the monotonicity of the exact solution which leads to a good approximation. To achieve this, we apply a roll-back operation to the matching sequence which guarantees the monotonic behavior as follows. After each Predictor phase (see above) of our algorithm, we compute the weight of the heaviest augmenting path of the current matching V_j. If this weight is smaller than the weight of the augmenting path that led to V_j we can continue with the Predictor phase. Otherwise we need to find a matching V_ℓ, $\ell < j$ where this property is fulfilled.

The algorithmic realization of this idea is described in Section 4.3 and pseudo code is given in Algorithm 3. For future reference we call this step of our algorithm the *Corrector* phase.

For the overall combinatorial Predictor-Corrector algorithm that produces a monotone approximation of (5) we refer to the pseudo code shown in Algorithm 1. A simple edge weighted graph is used in Figure 5 to demonstrate our Predictor-Corrector algorithm.

Note that the monotonicity property of our approximation is necessary to maintain the Persistency property of CVT. If we deal with a gradient vector field, then the critical points should be canceled in an order determined by the difference of their scalar values. The exact algorithm always cancels the pair of connected critical points with the smallest scalar difference. Our monotony preserving approximation algorithm guarantees that the scalar difference of canceled pairs always increases as we simplify the topological skeleton. As can be seen in Figure 6, this property is not only necessary to maintain Persistency but it also seems to be sufficient in practice.

4.3 Implementational Details

We now proceed by giving a detailed and accurate description of our approximation algorithm for (5) motivated in Section 4.2. To ensure a good reproducibility of our results this part will be quite technical. The main algorithm is given in Algorithm 1. The input of this algorithm is the link weighted simplicial graph $G = (U \dot{\cup} W, L, \omega)$, where $U \dot{\cup} W$ denotes a bipartition of the nodes S of the graph. For a detailed description of this input data we refer to Section 3.

The output of Algorithm 1 consists of an approximation of the heaviest maximum cardinality matching V_{k_n} and an array of maximum weight augmenting paths P. V_{k_n} and P enable us to efficiently reconstruct all matchings in the sequence (5). This is due to the fact the last path stored in P is an alternating path in V_{k_n} whose endpoints are both matched. We can therefore reconstruct V_{k_n-1} by taking the symmetric difference $\triangle$ of V_{k_n} with the last path stored in P. This operation can be interpreted as an inverse augmentation of the matching. Storing the sequence (5) in this fashion is much more efficient than storing the individual matchings. The size of an augmenting path is in $O(\sqrt{n})$ for our type of graphs (they are typically not space filling curves) while the size of a single matching is in $O(n)$.

Algorithm 1 Predictor-Corrector algorithm for (5)

```
Input: G
Output: $V_{k_n}$, P
1: M ← ∅
2: P ← ∅
3: loop
4:    (isFinished, M, P) ← Predict(G, M, P)
5:    if isFinished = false then
6:       (M, P) ← Correct(G, M, P)
7:    else
8:       $V_{k_n}$ ← M
9:       return ($V_{k_n}$, P)
```

We now describe the Predictor phase (Line 4 of Algorithm 1). The pseudo code for this procedure is given in Algorithm 2. Line 1 calls Algorithm 4 to compute for each node the heaviest augmenting path ending in it. The paths are stored implicitly via the node attribute *.predLink*, while the weight of the augmenting path is stored in the node attribute *.distance*. For a detailed description of Algorithm 4 see below. Line 2 computes the subset *A* of the nodes of partition *W* that are not covered by the matching and whose computed distance is finite. Line 8 extracts the augmenting path *p* that starts in the last element of *A* by following the node attribute *.predLink* through the graph and then removes the last element of *A*. Lines 9-12 check if *p* is still a valid augmenting path in the graph (the first augmenting path is always valid but may invalidate subsequent paths), augment the matching M along p by computing their symmetric difference $\triangle$ (see Figure 4) and append p to the list of augmenting paths *P*.

The Corrector phase (Line 6 of Algorithm 1) is similar to the Predictor phase described above. Its pseudo code is given in Algorithm 3. The procedure *weight* called in Lines 6,7 and 10 computes the weight of a given augmenting path, that is, the alternating sum of link weights.

Algorithm 2 Predictor phase

```
Input: G, M, P
Output: isFinished, M, P
1: (S.distance, S.predLink) ← BellmannFord(G, M)
2: A ← $\{s \in W \setminus S(M)$: s.distance $< \infty\}$
3: A ← sortByDistance(A)
4: if A = ∅ then
5:    return (true, M, P)
6: else
7:    while A ≠ ∅ do
8:       p ← getAugmentingPath(S, A.pop())
9:       if isValidAugmentingPath(M, p) then
10:         M ← M △ p
11:         P.push(p)
12:   return (false, M, P)
```

Algorithm 3 Corrector phase

```
Input: G, M, P
Output: M, P
1: loop
2:    (S.distance, S.predLink) ← BellmannFord(G, M)
3:    A ← $\{s \in W \setminus S(M)$: s.distance $< \infty\}$
4:    A ← sortByDistance(A)
5:    p ← getAugmentingPath(S, A.top())
6:    barrier ← weight(G, p)
7:    if barrier ≤ weight(G, P.top()) then
8:       return (M, P)
9:    else
10:      while barrier > weight(G, P.top()) do
11:         M ← M △ P.pop()
```

Almost all computational time of the overall approximation algorithm is spent in Algorithm 4. We therefore present its pseudo code and some details on the application specific changes we have introduced compared to the standard Bellman-Ford shortest path algorithm. The input of Algorithm 4 consists of the link weighted simplicial graph $G = (U\dot{\cup}W, L, \omega)$, the current matching M. The output consists of the weight of the heaviest augmenting path for each node in the graph stored in *.distance*, and the respective augmenting paths stored implicitly in *.predLink*.

Lines 1 - 16 initialize all variables so that the main Bellman-Ford loop computes the output described above. Note that Lines 9 - 15 correspond to the construction of the directed graph D_M described in Section 4.1. Lines 17 - 28 are a variant of Bellman-Ford optimized for the particular class of graphs we deal with, and modified to support an efficient parallelization, see Section 4.4. Instead of the *while* loop with an abort criterion one typically iterates Lines 22 - 25 n times. As the longest shortest path in our class of graphs is in $O(\sqrt{n})$ it is very beneficial to check whether we can abort this loop early. This is the purpose of Lines 16 - 18 and 28. Since the graph is not dense, only a small subset of nodes have to be considered in each iteration, which is achieved by the Lines 3, 20, 21 and 27.

Algorithm 4 Bellman-Ford matching variant

```
Input: G, M
Output: S.distance, S.predLink
1:  for all $s \in S$ do
2:      s.predLink ← nil
3:      isActive[s] ← true
4:      if $s \in U \setminus S(M)$ then
5:          s.distance ← 0
6:      else
7:          s.distance ← ∞
8:  $\vec{L} \leftarrow \emptyset$
9:  for all $\ell = \{u, w\} \in L, u \in U, w \in W$ do
10:     if $\ell \in M$ then
11:         $\vec{L} \leftarrow \vec{L} \cup (u, w)$
12:         $\vec{\omega}(\ell) \leftarrow \omega(\ell)$
13:     else
14:         $\vec{L} \leftarrow \vec{L} \cup (w, u)$
15:         $\vec{\omega}(\ell) \leftarrow -\omega(\ell)$
16: abort ← false
17: while abort = false do
18:     abort ← true
19:     for all $s \in S$ do
20:         if isActive[s] = true then
21:             isActive[s] ← false
22:             for all $\ell = (u, s) \in \vec{L}$ do
23:                 if s.distance > u.distance + $\vec{\omega}(\ell)$ then
24:                     s.distance ← u.distance + $\vec{\omega}(\ell)$
25:                     s.predLink ← $\ell$
26:                     for all $\ell = (s, w) \in \vec{L}$ do
27:                         isActive[w] ← true
28:                         abort ← false
```

4.4 Parallelization

As practically all computational time of Algorithm 1 is spent in the Bellman-Ford algorithm (Lines 16-28 of Algorithm 4), its efficiency is critical to the overall runtime. In [27] it is shown that an implementation of this shortest path problem using CUDA can result in a great performance increase. Unfortunately we cannot directly make use of this parallel approach to Bellman-Ford, as we need to know not only the distances, but also the shortest paths themselves in our application. Including the computation of the shortest paths via the variable *.predLink* in the algorithm presented in [27] results in a race condition that leads to invalid results.

We therefore need to formulate Bellman-Ford in a parallel fashion such that there are no race conditions. This can be achieved by iterating over the incoming edges (Line 22 of Algorithm 4) for each node of the graph instead of its outgoing edges as in [27]. We can then parallelize the loop in Line 19 of Algorithm 4 as the write accesses are exclusive for each thread (the Boolean array *isActive* and the Boolean variable *abort* may be written to concurrently but this does not pose a problem).

5 Results

5.1 Approximation Quality

To determine the quality of our approximation algorithm we applied it to a synthetic data set shown in Figure 6. The data set was produced by sampling the analytic function $f : [-1, 1]^2 \to \mathbb{R}$

$$f(x, y) = \sin(10\,x)\,\sin(10\,y)\,e^{-3\,(x^2+y^2)} \tag{6}$$

on a uniform triangulation with $16k$ vertices, adding uniform noise of the range $[-0.05, 0.05]$ to the sub domain $[0, 1] \times [-1, 1]$ and taking the gradient. Note that the spatially varying amount of noise provides a special challenge for the topological analysis. Figure 6 shows a visualization of this triangulated vector field as a surface line-integral-convolution (LIC) using the scalar value of f as the z-coordinate. We then solved (5) for this data set using the exact algorithm described in Section 4.1 and our new predictor-corrector algorithm described in Section 4.2. To evaluate the importance of the corrector phase, we also computed (5) using only the predictor phase. The 63 most important critical points, i.e. the critical points of V_{k_n-31}, are depicted for each algorithm in Figure 6.

The result of our predictor-corrector approximation algorithm (top-right) is very similar to the exact result (top-left). The predictor-only result (bottom-left) however is quite different from the exact result. This bad approximation behavior of the predictor-only algorithm can also be seen in the weight of the cancellations (see Section 3.2 for a definition) depicted in Figure 6, bottom-right. The x-axis represents the pair cancellations, while the y-axis shows the weight of the cancellations. The proposed predictor-corrector algorithm (blue curve) closely follows the exact algorithm (red curve) and is monotonically decreasing. In contrast, the predictor-only algorithm (green curve) is quite different from the exact algorithm and does not preserve the monotonic behavior of the exact solution.

5.2 Application

To demonstrate the usefulness of fast CVT we applied our algorithm to a real-world data set from climate research. This data set is a short subset of the IPCC AR4 climate projections, which were carried out at DKRZ by the Max-Planck-Institute for Meteorology with the coupled atmosphere-ocean model ECHAM5/MPI-OM. We have used the 10 meter wind components depicted by a surface LIC representation in Figure 8 using the pressure for the

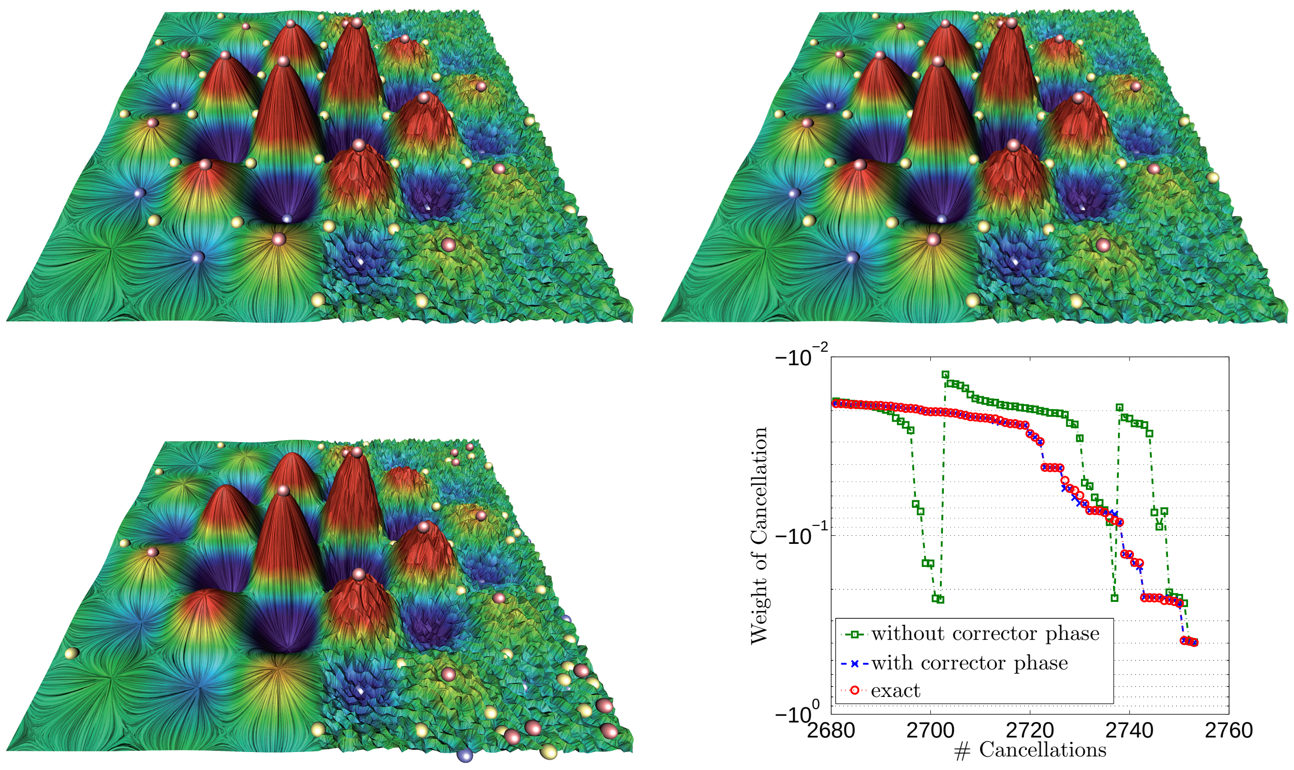

Fig. 6. Approximation vs. exact method. A synthetic data set, described in Section 5.1, is analyzed using the exact method (top-left), the novel predictor-corrector approximation algorithm (top-right), and the predictor-only variant (bottom-left). The 63 most dominant critical points are shown as blue (sinks), yellow (saddles), and red (sources) balls. The weights of the last cancellations are shown for the different algorithms in a semi-logarithmic plot (bottom-right).

color values. We sampled this data set on a simplicial graph with about 2.5 million nodes and computed (5) with our combinatorial predictor-corrector algorithm. The runtime using a CUDA implementation of our algorithm was 4 minutes. The estimated runtime using the exact algorithm described in [7] is about 6 weeks.

The full set of critical points of the initial combinatorial vector field V_{k_0} without any simplification is shown in Figure 7. The critical points in this Figure are scaled by their importance value given by the difference of the matching weights $\omega(V_j) - \omega(V_{j+1})$, see Section 3.2. Note that larger critical points correspond to strong pressure systems, even though the pressure values were not employed in the calculation of (5). This indicates the physical relevance of the persistence-like important measure induced by (5) for real-world vector field data.

5.3 Comparison

To compare the presented combinatorial approach to vector field topology with a continuous one [1], we analyzed the climate data set described above with both approaches. To compare the hierarchy of combinatorial vector fields to the single continuous extraction result, we selected the combinatorial vector field with the same number of critical points as the continuous extraction result. Both sets of critical points are depicted in Figure 8, the critical points extracted by the continuous method shown as black balls, the critical points computed by the combinatorial algorithm as white balls. Note that Figure 7 only seems to contain fewer critical points as they are scaled by their importance resulting in critical points that are smaller than a pixel.

As can be seen in Figure 8, most critical points of both methods coincide. The critical points found by the continuous method, that are not included in the combinatorial result, all appear in flat regions of the vector field, i.e. regions where the magnitude is close to zero. They are therefore not stable w.r.t. to perturbations of the data and may be considered as noise artifacts. Also, they may strongly depend on the chosen interpolation.

To describe the differences of our simplification strategy with the existing continuous extraction algorithms we can make use of the result shown in Figure 6. All existing simplification methods mentioned in Section 2 do not make use of the magnitude of the vector field. Therefore the peaks in the center of

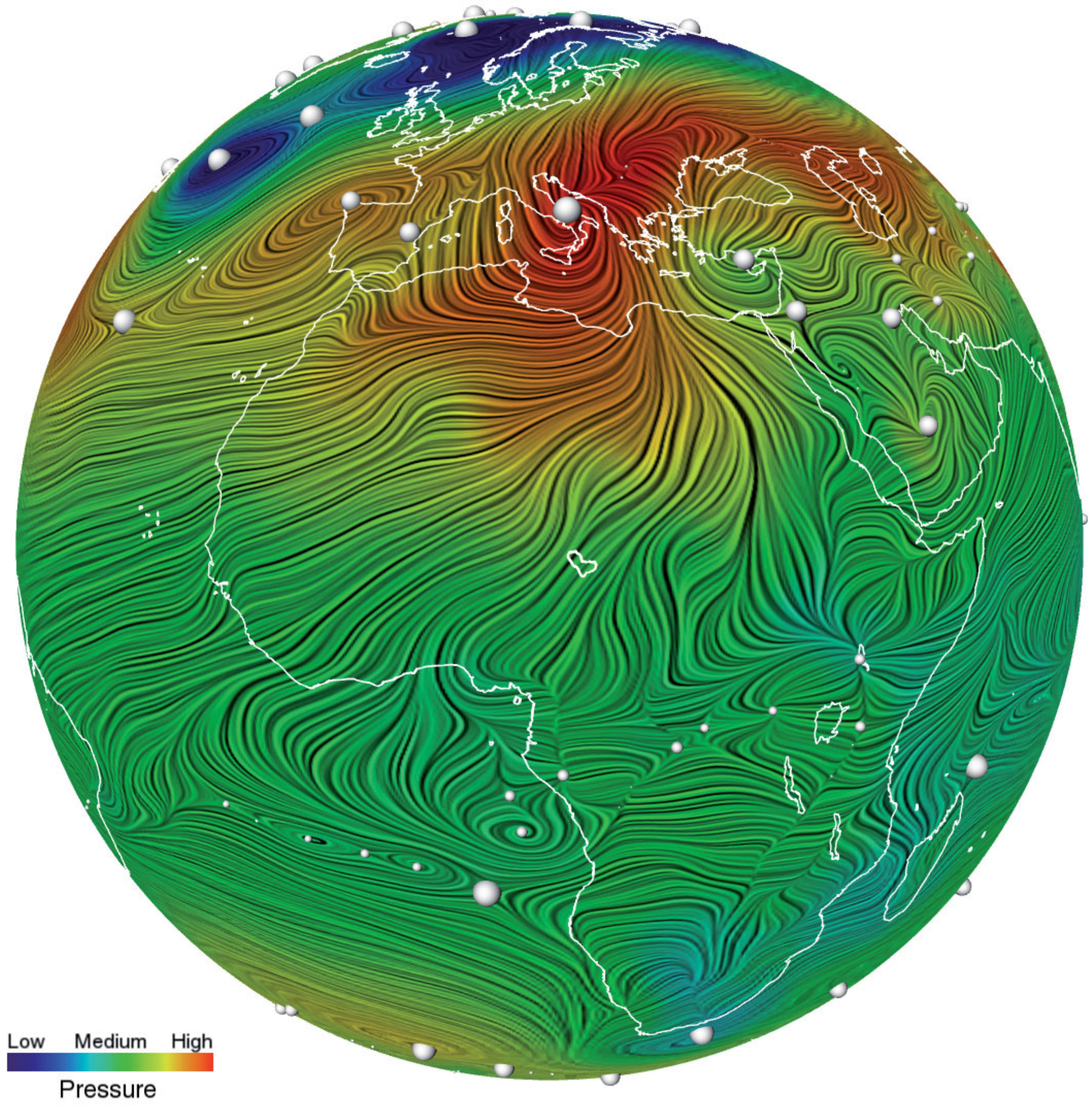

Fig. 7. Importance measure illustration. The 10 meter wind components form a data set from climate research are depicted as a surface LIC using the pressure for the color. The white balls show the critical points computed by the combinatorial method. The size of the balls is determined by the persistence like importance measure described in Section 3.2. Note that the pressure is only depicted here to illustrate the physical relevance of the importance measure.

the data set would be given the same importance as the small hill tops near the boundary of the data set. In contrast, the presented combinatorial approach takes the magnitude of the vector field into account, which results in the simplification hierarchy shown in Figure 3, right. The small hill tops near the boundary of the data set are canceled at an early stage, while the peaks in the center are canceled last.

5.4 Performance

To measure the performance advantage of our algorithm over the exact algorithm we computed (5) for the aneurysm data used in [7], the synthetic data set described in Section 5.1, and four resolutions of a real-world data set from climate research described in Section 5.3 using the exact algorithm described in Section 4.1 and our approximation algorithm. The timings for an Intel Core 2 Duo 3 GHz CPU with a Nvidia Geforce GTX 260 Core 216

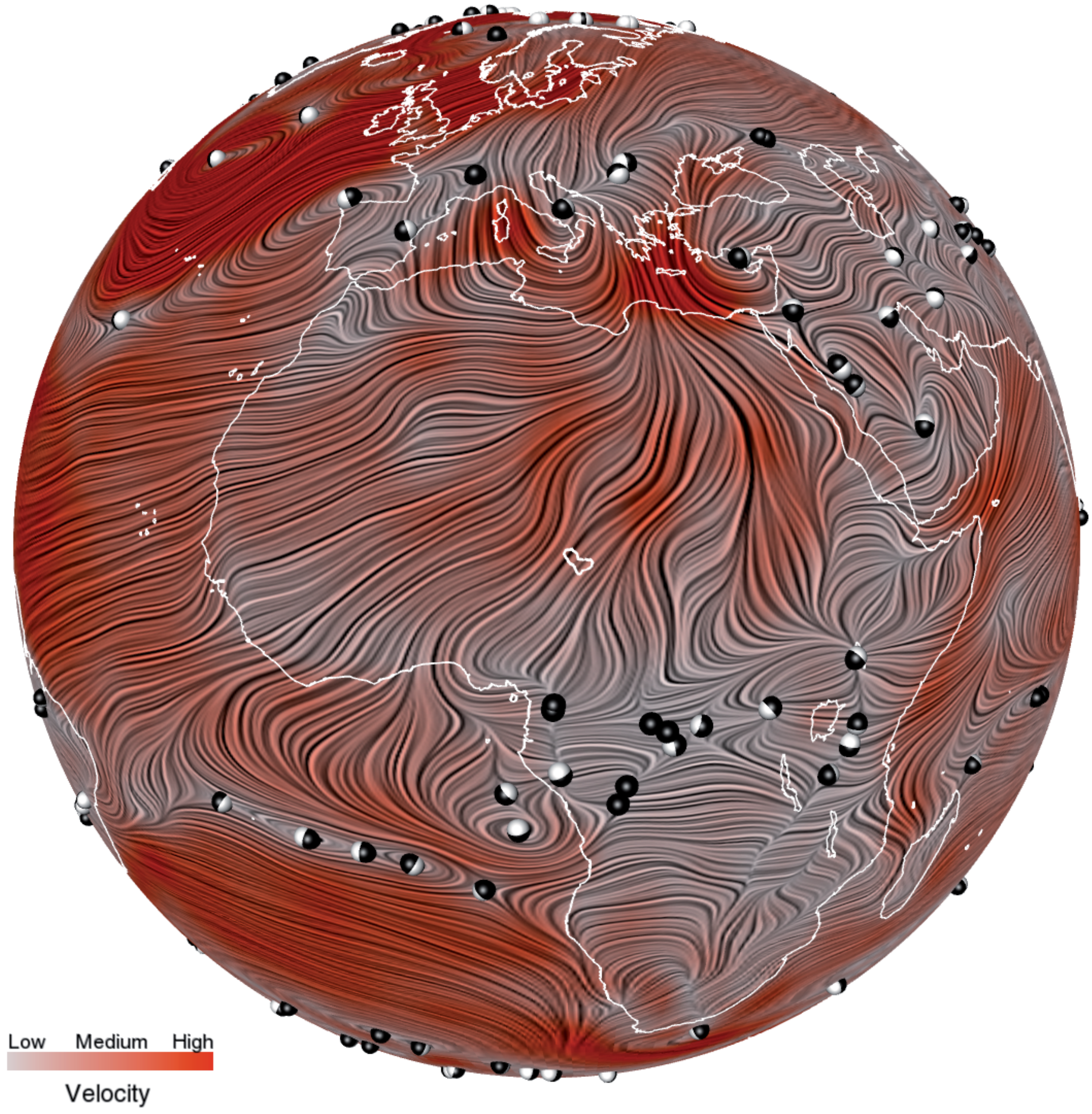

Fig. 8. Comparison with continuous approach. The 10 meter wind components form a data set from climate research are depicted as a surface LIC using the wind velocity for the color. The black balls show the critical points of the vector field computed by a continuous method, while the white balls show an extraction result of the presented combinatorial approach.

graphics card are given in Table 1. To determine the parallel scalability we computed all data sets with 1 thread, 2 threads, and on the GPU. The CPU version was implemented with OpenMP, while the GPU version was implemented in CUDA. Due to the extremely long runtime of the exact method we have not measured its runtime for the larger data sets.

In theory, Algorithm 1 could have a very large computational complexity - even an upper bound for the complexity is hard to derive due to the Predictor-Corrector interplay. In practice however, our approximation algorithm has an empirical complexity of $O(n^{3/2} \log n)$ as can be seen in the $1\times$CPU column of Table 1. The OpenMP implementation shows a near perfect parallel scaling when going from one thread to two threads. The speed up provided by the CUDA implementation ranges between $1.5\times$ and $30\times$ - the larger the data set the bigger the speed up.

TABLE 1
Runtime analysis. (5) was computed for various data sets with the exact algorithm used in [7] and our novel algorithm using an OpenMP implementation (CPU) and a CUDA implementation (GPU).

Name	#nodes	[7]	1×CPU	2×CPU	GPU
Aneurysm	60k	2360s	12s	6s	4s
Synthetic	97k	6258s	31s	16s	8s
Climate 1	154k	15919s	55s	28s	10s
Climate 2	614k	*	569s	284s	52s
Climate 3	2458k	*	4982s	2619s	237s
Climate 4	9830k	*	37602s	19014s	1220s

6 Discussion

Our novel approximation algorithm for the fundamental graph problem (5) of combinatorial vector field topology alleviates the main weakness of the computational framework presented in [7], while it preserves the three main advantages of CVT mentioned in Section 1.

As demonstrated in Section 5.1 the *Persistency* property of CVT is maintained by our approximation algorithm. Each point is given an importance value that indicates its relevance in the overall data set as discussed in Section 5.2. Since the approximation algorithm results in a matching sequence, the resulting topological skeletons are always *consistent* with the topology of the domain. As our predictor-corrector algorithm does not introduce any computational parameters, the *Simplicity* of CVT is also maintained.

The main weakness of CVT is its large computational runtime. This weakness is alleviated by our algorithm that reduces the runtime from weeks to minutes for large data sets, see Table 1. While this algorithm is only approximative in nature, it produces results that are visually indistinguishable from the exact solution of (5), see Section 5.1. The algorithm is also capable of dealing with data sets with varying amounts of noise, as can be seen in Figure 6.

Overall, our algorithm significantly broadens the applicability of combinatorial vector field topology.

Our predictor-corrector approximation algorithm for (5) produces results that are very close to the exact solution for the graphs we consider. We assume that this is due to the preservation of the monotony and the special structure of the simplicial graph and the symmetries of the link weights (2). From a graph theoretic point of view it would be interesting to find out whether there are other classes of graphs where our algorithm for (5) produces such good results. A possible limitation for the application of our algorithm to other graph problems is the fact that the Bellman-Ford algorithm does not work when the graph contains a circle with negative weight [22]. In the exact method we can use Bellman-Ford due to the maximality property (4) of the computed matchings (see Section 4.1). In our approximate setting this maximality property does not necessarily hold and there may theoretically exist circles of negative weight. However in all of our experiments we have never encountered such a case. This may be due to the monotonic behavior of our approximation or the manifold structure of the graphs. A thorough theoretical investigation of this empirical observation may be worthwhile.

The main remaining problem of CVT that limits its applicability is the geometric embedding of the separatrices and periodic orbits as shown and discussed in [7]. We hope that the presented algorithm might be useful in alleviating this problem as it allows for a quick evaluation of possible remedies and enables the use of much finer meshes to represent the input data.

The presented algorithm might enable an extension of CVT to 3D - computing (5) for a 2D simplicial graph with 10 million nodes takes only about 20 minutes, see Table 1. It is however unclear how the complexity of the algorithm would be affected by the different structure of a 3D simplicial graph. Another problem in 3D may be the rather large memory consumption of the explicit representation of the simplicial graph and the fact that the extraction of topological features in 3D is a lot more involved.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments that significantly improved the quality of this paper. This work was funded by the DFG Emmy-Noether research program. Finally, we would like to thank David Günther, Jens Kasten, Daniel Baum, and Tino Weinkauf for many fruitful discussions on this topic.

References

[1] T. Weinkauf, “Extraction of topological structures in 2d and 3d vector fields,” Ph.D. dissertation, University Magdeburg, 2008. [Online]. Available: http://www.zib.de/weinkauf/

[2] X. Tricoche, G. Scheuermann, H. Hagen, and S. Clauss, “Vector and tensor field topology simplification on irregular grids,” in *VisSym ’01: Proceedings of the symposium on Data Visualization 2001*, D. Ebert, J. M. Favre, and R. Peikert, Eds. Wien, Austria: Springer-Verlag, May 28–30 2001, pp. 107–116.

[3] X. Tricoche, G. Scheuermann, and H. Hagen, “Continuous topology simplification of planar vector fields,” in *VIS ’01: Proceedings of the conference on Visualization ’01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 159–166.

[4] T. Weinkauf, H. Theisel, K. Shi, H.-C. Hege, and H.-P. Seidel, “Extracting higher order critical points and topological simplification of 3D vector fields,” in *Proc. IEEE Visualization 2005*, Minneapolis, U.S.A., October 2005, pp. 559–566.

[5] T. Klein and T. Ertl, "Scale-space tracking of critical points in 3d vector fields," in *Topology-based Methods in Visualization*, ser. Mathematics and Visualization, H. H. Helwig Hauser and H. Theisel, Eds. Springer Berlin Heidelberg, May 2007, pp. 35–49.
[6] G. Chen, K. Mischaikow, R. Laramee, P. Pilarczyk, and E. Zhang, "Vector field editing and periodic orbit extraction using morse decomposition," *IEEE Transactions in Visualization and Computer Graphics*, vol. 13, pp. 769–785, 2007.
[7] J. Reininghaus and I. Hotz, "Combinatorial 2d vector field topology extraction and simplification," in *TopoInVis '09*, submitted for publication and presented 2009.
[8] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological persistence and simplification," *Discrete Comput. Geom.*, vol. 28, pp. 511–533, 2002.
[9] H. Edelsbrunner and J. Harer, "Persistent homology — a survey," in *Surveys on Discrete and Computational Geometry: Twenty Years Later*, J. E. Goodman, J. Pach, and R. Pollack, Eds. AMS Bookstore, 2008, vol. 458, pp. 257–282.
[10] J. Helman and L. Hesselink, "Representation and display of vector field topology in fluid flow data sets," *Computer*, vol. 22, no. 8, pp. 27–36, Aug. 1989.
[11] T. Wischgoll and G. Scheuermann, "Detection and visualization of closed streamlines in planar flows," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 2, pp. 165–172, 2001.
[12] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel, "Grid-independent detection of closed stream lines in 2d vector fields," in *Proceedings of the VMV Conference 2004*, Stanford, USA, November 2004, p. 665.
[13] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post, "Topology-based flow visualization, the state of the art," in *Topology-based Methods in Visualization*, ser. Mathematics and Visualization, H. H. Helwig Hauser and H. Theisel, Eds. Springer Berlin Heidelberg, May 2007, pp. 1–19.
[14] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Morse-smale complexes for piecewise linear 3-manifolds," in *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*. New York, NY, USA: ACM, 2003, pp. 361–370.
[15] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann, "A topological approach to simplification of three-dimensional scalar functions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 474–484, 2006.
[16] A. Gyulassy, "Combinatorial construction of morse-smale complexes for data analysis and visualization." Ph.D. dissertation, University of California, Davis, 2008.
[17] T. Lewiner, H. Lopes, and G. Tavares, "Applications of forman's discrete morse theory to topology visualization and mesh compression," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 5, pp. 499–508, 2004.
[18] R. Forman, "A user's guide to discrete morse theory," in *Proceedings of the 2001 Internat. Conf. on Formal Power Series and Algebraic Combinatorics*, ser. Advances in Applied Mathematics, 2001. [Online]. Available: citeseer.ist.psu.edu/forman01users.html
[19] T. Lewiner, "Geometric discrete Morse complexes," Ph.D. dissertation, Department of Mathematics, PUC-Rio, 2005, advised by Hlio Lopes and Geovan Tavares. [Online]. Available: http://www.matmidia.mat.puc-rio.br/∼tomlew/phd_thesis_puc_uk.pdf
[20] R. Forman, "Morse theory for cell complexes," *Advances in Mathematics*, vol. 134, pp. 90–145, 1998.
[21] ——, "Combinatorial vector fields and dynamical systems," *Mathematische Zeitschrift*, vol. 228, pp. 629–681, 1998.
[22] A. Schrijver, *Combinatorial Optimization*, R. Graham, B. Korte, L. Lovasz, A. Widgerson, and G. Ziegler, Eds. Springer, 2003.
[23] S. Hougardy and D. Drake, "Approximation algorithms for the weighted matching problem," in *Oberwolfach Report 28*, 2004.
[24] A. Hatcher, *Algebraic Topology*. Cambridge, U.K.: Cambridge University Press, 2002, available at http://www.math.cornell.edu/∼hatcher/AT/ATpage.html.
[25] J. Reininghaus, D. Günther, I. Hotz, S. Prohaska, and H.-C. Hege, "A computational framework for data analysis using discrete morse theory," in *3rd Int. Congress on Mathematical Software - ICMS 2010,*, Kobe, Japan, Sept. 13 - 15, 2010, accepted for publication.
[26] H. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
[27] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *Proc of IEEE International Conference on High Performance Computing*, 2007.

Jan Reininghaus Jan Reininghaus studied mathematics with a focus on numerical analysis at the Humboldt University of Berlin, Germany. He received the M.S. degree in Mathematics in 2007. He is a main author of OpenFFW, an open source finite element framework written in Matlab.

Christian Löewen Christian Löewen studied computer science with focus on hardware oriented programming at the University of applied science, Iserlohn, Germany. During his employment as a student research assistant at the Scientific Visualization department at the Zuse Institute in Berlin he was involved in the development process of the CVT framework. The focus of his work was the parallel formulation of computationally intensive algorithms. He wrote his diploma thesis on the implementation and optimization of these algorithms for the CUDA framework. In 2009 he received his diploma(UAS) and now works as a software developer in Aachen, Germany.

Ingrid Hotz Ingrid Hotz received the M.S. degree in theoretical Physics from the Ludwig Maximilian University in Munich Germany and the PhD degree from the Computer Science Department at the University of Kaiserslautern, Germany. During 2003 – 2006 she worked as a postdoctoral researcher at the Institute for Data Analysis and Visualization (IDAV) at the University of California. Currently she is the leader of a junior research group at the Zuse Institute in Berlin Germany. Her research interests are in the area of data analysis and scientific visualization with focus on tensor and vector fields.