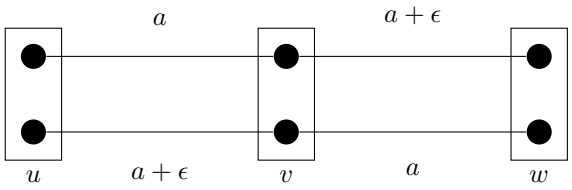


Figure 5. Example of non-integer solution to the Local Polytope relaxation for a chain graphical model containing nodes $\{u, v, w\}$ with 2 labels for each node and containing only the bottleneck pairwise potentials $\phi_{uv}(\cdot,\cdot), \phi_{vw}(\cdot,\cdot)$.

6. Appendix

Lemma 1. *The following relaxation over the local polytope Λ (4) is not tight for bottleneck labeling problem:*

$$\min_{\substack{\mu\in\Lambda\\ b}} \sum_{i\in V} \langle\, \theta_i, \mu_i \,\rangle + \sum_{ij\in E} \langle\, \theta_{ij}, \mu_{ij} \,\rangle + b \tag{15}$$

s.t.

$$b \geq \langle\, \phi_i, \mu_i \,\rangle, \qquad \forall i \in V \tag{16a}$$
$$b \geq \langle\, \phi_{ij}, \mu_{ij} \,\rangle, \qquad \forall ij \in E \tag{16b}$$

Proof. We give a proof by example as in the Figure 5 with a 3 node binary graphical model only containing pairwise bottleneck potentials with $\epsilon > 0$. Both integer optimal solutions have cost: $max(a, a+\epsilon) = a+\epsilon$, whereas the optimal solution of the LP relaxation $\mu_{uv}(0,0) = \mu_{uv}(1,1) = \mu_{vw}(0,0) = \mu_{vw}(1,1) = 0.5$ has cost $a + \frac{\epsilon}{2}$. □

Lemma 2. Shortest path computation (line 6) on directed acyclic graph D can be done in linear time in the number of arcs A using breadth-first traversal by Algorithm **dsp_chain**. However, shortest path update is performed every time after introducing each arc in A (lines 4-10), thus making the runtime $\mathcal{O}(|A|^2)$ i.e. quadratic in the number of arcs in A. The sorting operation adds an additional factor of $|A| \log |A|$. □

6.1. Experiment details

Cost formulation: For unary potentials of seed node s, we set $\theta_s(z_s) = \begin{cases} 0, & z_s = z_s^* \\ \infty, & \text{otherwise} \end{cases}$. Rest of the unary potentials are computed as:

$$\theta_i(z_i) = -\log(\text{CNN}_f(I_s(z_s^*), I_i(z_i))) \tag{17}$$

The pairwise potentials contain terms in-addition to patch similarity from the CNN. First, we compute optical flow using the gradient structure tensor based approach of [1] on the seismic volume along the x and y-axis resulting in optical flow u_x and u_y. From this we compute the displacement penalization for edges in x-direction as $f_{ij}(z_{ij}) =$

chain_to_dag: chain MRF to directed acyclic graph transformation

Data: Chain MRF: $(G = (V, E), \mathcal{X}, \{\theta_f\}_{f\in V\cup E})$,
Bottleneck potentials: $\{\phi_f(x_f)\}_{f\in V\cup E}$,
Result: Directed graph: $D = (W, A)$
Linear costs: $\sigma(h,t) \quad \forall (h,t) \in A$
Bottleneck costs: $\omega(h,t) \quad \forall (h,t) \in A$
Start and terminal nodes s, t

`// Represent each label with two nodes and add source, sink:`

1 $W = \{s,t\} \cup \{x_i, \overline{x}_i : i \in V, x_i \in \mathcal{X}_i\}$;

`// Add arcs to represent potentials:`

2 $A = \begin{array}{l} \{(s, x_1) : x_1 \in \mathcal{X}_1\} \cup \\ \{(x_i, \overline{x}_i) : i \in V, x_i \in \mathcal{X}_i\} \cup \\ \{(\overline{x}_i, x_{i+1}) : j \in [n-1], x_{i,i+1} \in \mathcal{X}_{i,i+1}\} \cup \\ \{(\overline{x}_n, t) : x_n \in \mathcal{X}_n\} \end{array}$

`// Unary potentials:`

3 $\begin{array}{l} \sigma(\overline{x}_i, x_i) = \theta_i(x_i) \\ \omega(\overline{x}_i, x_i) = \phi_i(x_i) \end{array} \qquad \forall i \in V, x_i \in \mathcal{X}_i;$

`// Pairwise potentials:`

4 $\begin{array}{l} \sigma(\overline{x}_i, x_{i+1}) = \theta_{i,i+1}(x_{i,i+1}) \\ \omega(\overline{x}_i, x_{i+1}) = \phi_{i,i+1}(x_{i,i+1}) \end{array}$
$\forall (i, i+1) \in E, x_{i,i+1} \in \mathcal{X}_{i,i+1};$

`// Connect s and t:`

5 $A' = \cup \begin{array}{l} \{(s, x_1) : x_1 \in \mathcal{X}_1\} \\ \{(\overline{x}_n, t) : x_n \in \mathcal{X}_n\} \end{array}$;

dsp_chain: dynamic shortest path on chains

Data: Diagraph: $D = (W, A)$
Node distances: $d(w), \quad \forall w \in W$
Arc costs: $\sigma(p,q), \quad \forall (p,q) \in A$
Arc to insert: $(u,v), \quad u, v \in W$
Result: Updated node distance: d
Updated nodes: $S \subseteq W$

1 Initialize $Q = \varnothing$;
2 $Enqueue(Q, (u,v))$;
3 **while** $Q \neq \varnothing$ **do**
4 $\quad (p,q) = Dequeue(Q)$;
5 $\quad$ **if** $d(q) \geq c(p,q) + d(p)$ **then**
6 $\quad\quad$ **continue**;
7 $\quad$ **end**
8 $\quad d(q) := c(p,q) + d(p)$;
9 $\quad S = S \cup \{q\}$;
$\quad$ `// Enqueue outgoing arcs from q for possible distance update:`
10 $\quad$ **foreach** $(q,r) \in A$ **do**
11 $\quad\quad Enqueue(Q, (u,v))$;
12 $\quad$ **end**
13 **end**

$|z_i - z_j - u_x(x_i, y_i, z_i)|$, and analogously for edges in y-direction. Next, we compute a coherence estimate C_{ij} [1] indicating whether the optical flows u_x and u_y are reliable. The pairwise MRF potentials combine the above terms into a sum of appearance terms (14) and weighted discontinuity penalizers:

$$\theta_{ij}(z_{ij}) = \begin{array}{l} -\log(\text{CNN}_n(I_i(z_i), I_j(z_j)) \\ +C_{ij}(z_{ij})|f_{ij}(z_{ij})| \end{array} \tag{18}$$

The second term allows discontinuities in the horizon surfaces where orientation estimates can be incorrect and penalizes discontinuities where the horizon is probably continuous. The authors from [44, 13] do not use the coherence estimate, making their cost functions less robust. The bottleneck pairwise potentials are:

$$\phi_{ij}(z_{ij}) = |E|\ \theta_{ij}(z_{ij}) \tag{19}$$

The scaling parameter $|E|$ in (19) makes the bottleneck potential invariant to the grid size.

CNN training: We use six out of eleven labeled horizons for training the CNN adopted from [45] mentioned in Figure 6. To prevent the CNN from learning the whole ground truth for these six horizons only 10% of the possible patches are used. For data augmentation, translation is performed on non-matching patches as also done in [38] for stereo. The validation set contains 2% of the possible patches from each of the eleven horizons.

For training the CNN_n in (18), edges in underlying MRF models are sampled for creating the patches. Training was done using PyTorch [26] for 200 epochs and the model with the best validation accuracy (95%) was used for computing the potentials.

For training the CNN_f used in (17), we use the same procedure as above except that the patches are sampled randomly and thus their respective nodes do not need to be adjacent. The best validation accuracy on the trained model was 83%.

6.2. Primal rounding details

The function $s(i)$ defines some ordering for all nodes i in V. Based on this order, we sequentially fix labels of the respective node as in Algorithm **primal_rounding**.

For any given dual variables, there is a whole set of equivalent dual variables that give the same dual lower bound. However, when rounding with **primal_rounding**, the final solution is sensitive to the choice of dual equivalent dual variables. One of the reasons is that the rounding is done on the MRF subproblems only, so we want to ensure that the dual variables λ carry as much information as possible. Therefore, we propose a propagation mechanism that modifies the dual variables $\boldsymbol{\eta}$ of the bottleneck potentials

primal_rounding: primal rounding based on MRF subproblem [19]

Data: MRF: $(G = (V, E), \mathcal{X}, \boldsymbol{\lambda})$,
Ordering of nodes in G: $s(v), \quad \forall v \in V$.
Result: Primal labeling: $\hat{\boldsymbol{x}} \in \mathcal{X}_V$.

1 $V_o \coloneqq V$;
2 Sort nodes in V_o w.r.t ordering $s(v)$;
3 $V_l \coloneqq \varnothing$, `// Set of labeled nodes`
4 **for** $i \in V_o$ **do**
`/* Assign label to node i in accordance with the nodes already labeled: */`
5 $\hat{x}_i \coloneqq \arg\min_{x_i \in \mathcal{X}_i} \left[\lambda_i(x_i) + \sum_{j \in V_l : ij \in E} \lambda_{ij}(x_i, \hat{x}_j) \right]$;
6 $V_l = V_l \cup \{i\}$, `// Mark i as labeled`
7 **end**

such that the overall lower bound is not diminished, while at the same time making the MRF dual variables as informative as possible. This procedure is described in Algorithm **Min_Marginals_BMRF**.

Schemes similar to Algorithm **Min_Marginals_BMRF** were also proposed in [30] for exchanging information between pure MRF subproblems. Intuitively, the goal in such schemes is to extract as much information out of a source subproblem (in our case bottleneck MRF subproblem) and send it to the target subproblem (in our case MRF subproblem) such that the overall lower bound does not decrease. Such a strategy helps the target subproblem in taking decisions which also comply with the source subproblem.

Algorithm **Min_Marginals_BMRF** computes min-marginals of nodes for a given bottleneck chain subproblem u. Proceeding in a similar fashion as Algorithm **chain_bottleneck**, min-marginals computation on a chain u is done as follows:

1. As an input, the costs of solution in all other chains excluding u are required, which can be obtained in the similar fashion as was done before in Algorithm **chain_decomp** (lines 1-10).

2. The algorithm maintains forward and backward shortest path distances d_r, d_l (i.e., distances from source, sink resp.). This helps in finding the cost of minimum distance path passing through any given node in the directed acyclic graph of chain u.

3. Similar to Algorithm **chain_bottleneck**, bottleneck potentials are sorted and the bottleneck threshold is relaxed minimally on each arc addition.

4. On every arc addition, the set of nodes for which distance from source/sink got decreased are maintained in S_r, S_l resp. Only this set of nodes will need to re-compute their min-marginals.

Min_Marginals_BMRF: Min marginals for bottleneck MRF subproblems

Data: Bottleneck chain graphs: $\{\mathbb{G}_l\}_{l\in[k]}$,
Linear potentials on chains: $\{\boldsymbol{\eta}^l\}_{l\in[k]}$,
Bottleneck potentials on chains: $\{\boldsymbol{\phi}^l\}_{l\in[k]}$,
Chain to compute min-marginals: $u \in [k]$,
Costs in higher level graph $\overline{H} = ([k] \setminus \{u\}, \varnothing)$:
Bottleneck costs: $\overline{b}$, Linear costs: $\overline{c}$

Result: Min-marginals of nodes in u: $m_i(\overline{y}_i^u) =$

$$\min_{b\in B}\left[\begin{array}{l}\zeta(b) + \min\limits_{\substack{y^u\in Y^u(b):\\ \overline{y}_i^u = y_i^u}} \langle\, \boldsymbol{\eta}^u, \boldsymbol{y}^u \,\rangle \\ + \sum\limits_{l\in[k]\setminus\{u\}} \min\limits_{y^l\in Y^l(b)} \langle\, \boldsymbol{\eta}^l, \boldsymbol{y}^l \,\rangle\end{array}\right], \quad \forall i \in \mathbb{V}_i, \overline{y}_i^u \in \mathcal{X}_i$$

`// Initialize min-marginals:`

1 $m_i(y_i) \coloneqq \infty, \quad \forall i \in \mathbb{V}_u, y_i \in \mathcal{X}_i$;

`// Represent chain` u `as DAG:`

2 $(D = (W, A), \boldsymbol{\sigma}, \boldsymbol{\omega}) \leftarrow$ **chain_to_dag**$(\mathbb{G}_u, \boldsymbol{\eta}^u, \boldsymbol{\phi}^u)$;

`// Forward (source-to-sink) shortest path structures:`

3 $A'_r = \cup \begin{array}{l}\{(s, x_1) : x_1 \in \mathcal{X}_1\} \\ \{(\overline{x}_n, t) : x_n \in \mathcal{X}_n\}\end{array}$

4 $d_r(s) \coloneqq 0, d_r(w) \coloneqq \infty, \forall w \in W \setminus \{s\}$;

`// Backward (sink-to-source) shortest path structures:`

5 $A'_l = \cup \begin{array}{l}\{(x_1, s) : x_1 \in \mathcal{X}_1\} \\ \{(t, \overline{x}_n) : x_n \in \mathcal{X}_n\}\end{array}$

6 $d_l(t) \coloneqq 0, \quad d_l(w) \coloneqq \infty, \forall w \in W \setminus \{t\}$;

7 Sort A according to values $\boldsymbol{\omega}$;

8 **for** $(n_i, n_j) \in A$ *in ascending order* **do**

9 $A' = (n_i, n_j) \cup A'$;

`// Forward update:`

10 $(d_r, S_r) =$ **dsp_chain**$((W, A'_r), d_r, \boldsymbol{\sigma}, (n_i, n_j))$;

`// Backward update:`

11 $(d_l, S_l) =$ **dsp_chain**$((W, A'_l), d_l, \boldsymbol{\sigma}, (n_j, n_i))$;

12 **for** $\{n = (v, y_v)\} \in S_r \cup S_l$ **do**

`// Update min-marginal of node` $v \in \mathbb{V}_u$, `label` $y_v \in \mathcal{X}_v$:

13 $(b_v, c_v) \leftarrow$ **unary_bottleneck**(

14 $(\{\overline{H}, n\}, \varnothing), \{\overline{b}, \omega(n))\}, \{\overline{c}, d_r(n) + d_l(n)\})$;

15 $m_v(y_v) \coloneqq \min\left(m_v(y_v), \zeta(b_v) + c_v\right)$;

16 **end**

17 **end**

The above-mentioned Algorithm **Min_Marginals_BMRF** only calculates the min-marginals for nodes, the calculation of min-marginals for edges can be carried out in similar way by also updating those edges of the DAG whose adjacent node gets updated (lines 14- 15). After computing the min-marginals, messages to MRF tree Lagrangians $\boldsymbol{\lambda}$ can be computed by appropriate normalization.

6.3. Results

Figures 7- 17 contain the visual comparison of tracked horizon surfaces mentioned in Table 1. Similar to Figure 4, the color of the surface denotes depth.

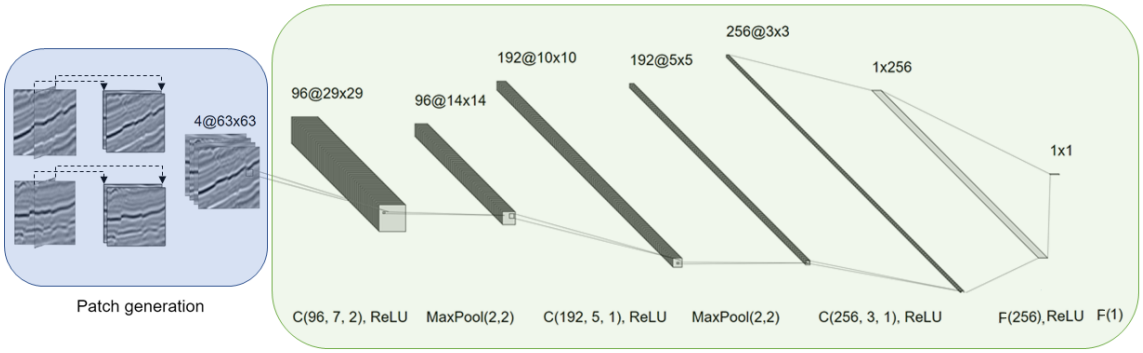
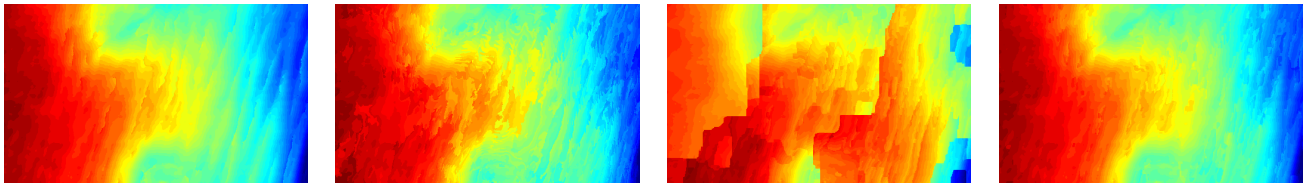
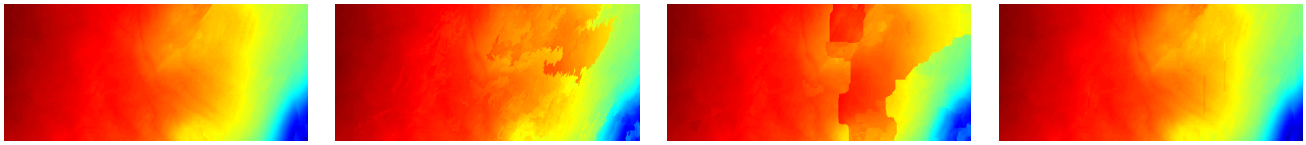


Figure 6. Workflow to compute the patch matching probabilities (14). Two axis-aligned patches are extracted around each voxel in seismic volume to get 2-channel patch image. These two images are concatenated to get a 4-channel image which would be used as an input to the CNN [45] for computing patch matching probabilities by softmax activation at the output. C(n, k, s) denotes a convolutional layer having n filters of size $k \times k$ with stride s, F(n) to a fully connected layer with n outputs, and MaxPool(m, n) to max-pooling with kernel size $m \times m$ and stride n.



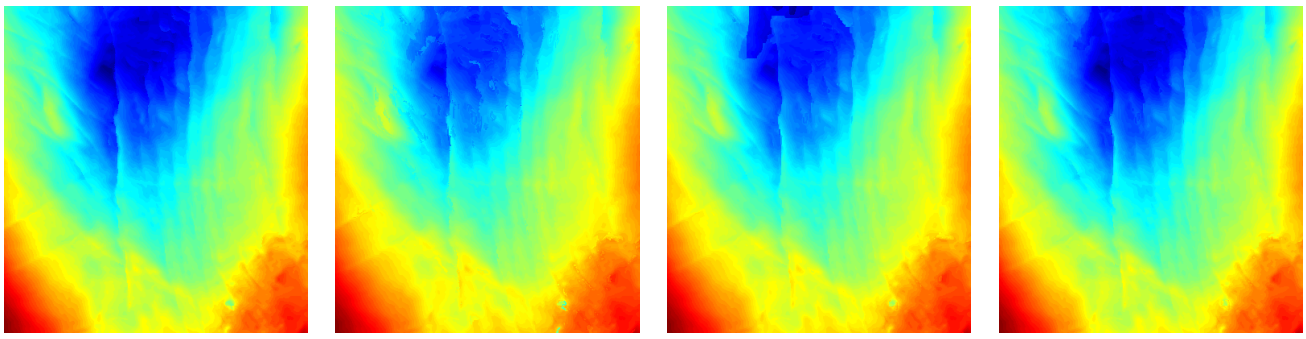
(a) Ground-truth (b) `MST` (c) `MRF` (d) `B-MRF`

Figure 7. F3-Netherlands-II



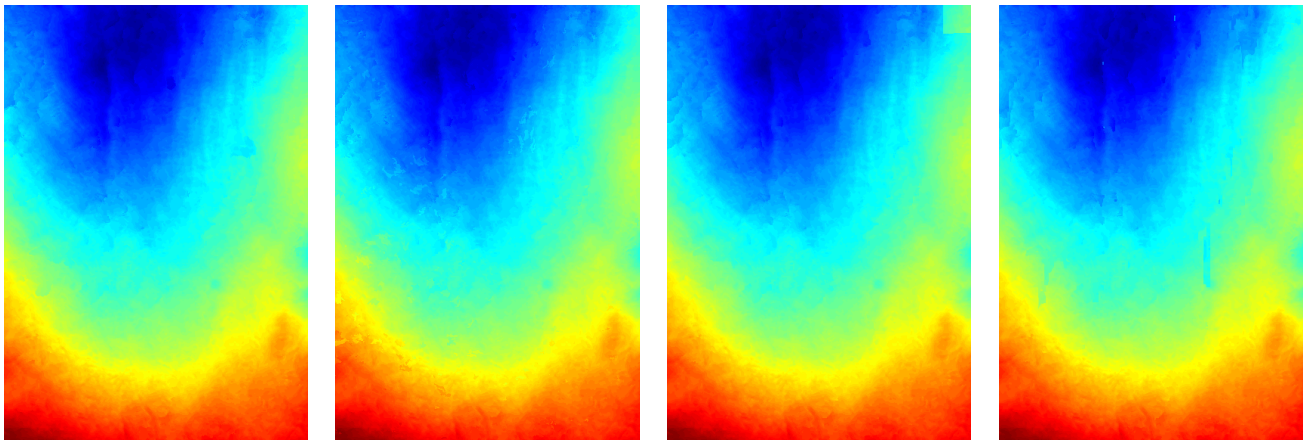
(a) Ground-truth (b) `MST` (c) `MRF` (d) `B-MRF`

Figure 8. F3-Netherlands-III



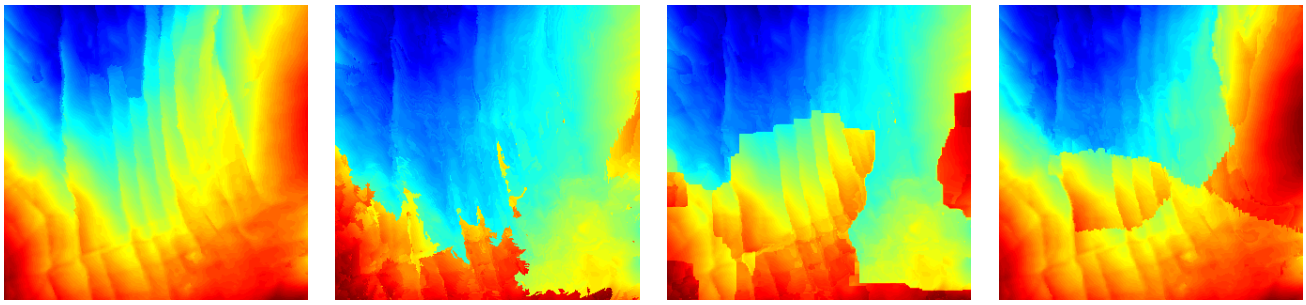
(a) Ground-truth (b) `MST` (c) `MRF` (d) `B-MRF`

Figure 9. F3-Netherlands-IV



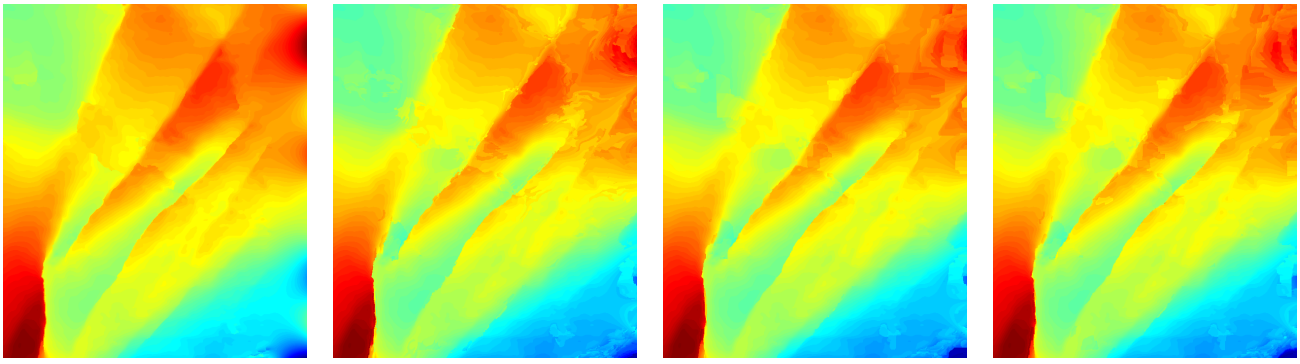
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 10. F3-Netherlands-V



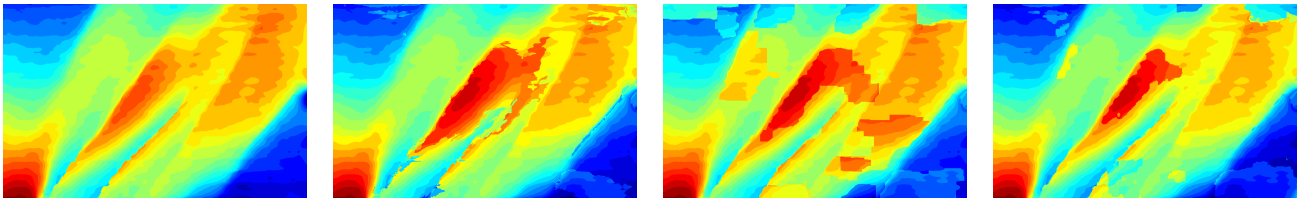
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 11. F3-Netherlands-VI



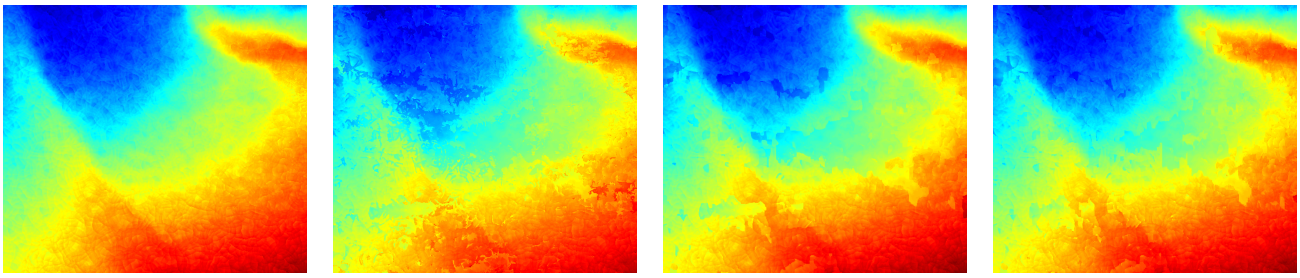
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 12. Opunaka-3D-I



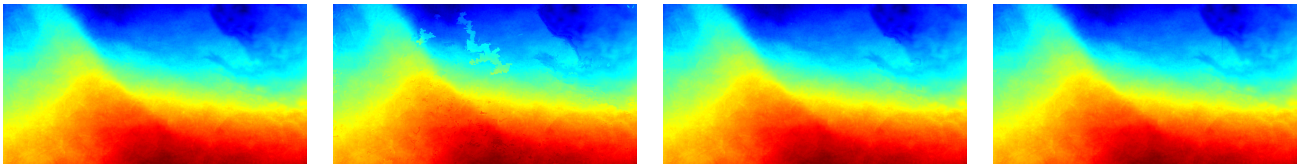
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 13. Opunaka-3D-II



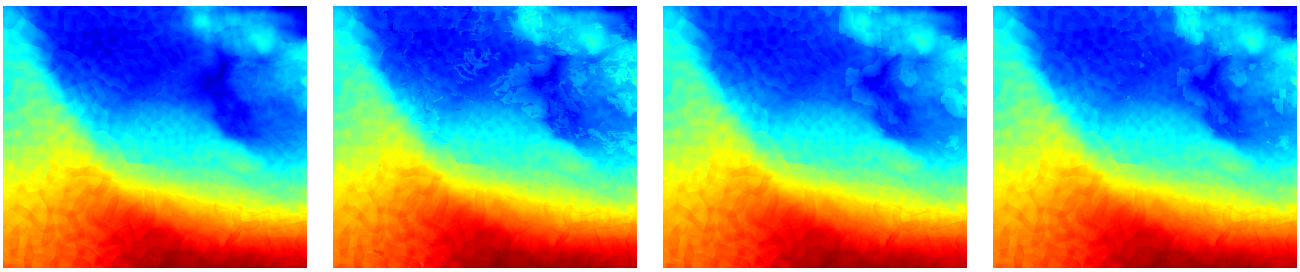
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 14. Waka-3D-I



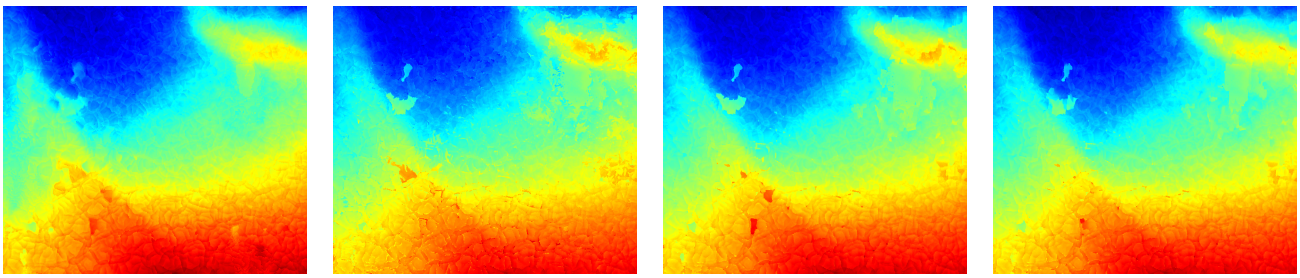
(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 15. Waka-3D-II



(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 16. Waka-3D-III



(a) Ground-truth (b) MST (c) MRF (d) B-MRF

Figure 17. Waka-3D-IV