Supplementary Material for "Exploiting Spatial Invarance for Scalable Unsupervised Object Tracking"

A Module Details

A.1 Propagation

The majority of the propagation module was described in Section 3.4. Here we provide a few additional details.

Attribute Updates. In Section 3.4 we described functions f^{where}, f^{what} and f^{depth} which perform propagation updates:

$$f^{y/x}(o^{y/x}_{(t-1)}, \tilde{z}^{y/x}) = o^{y/x}_{(t-1)} + \tanh(\tilde{z}^{y/x})$$

$$f^{h/w}(o^{h/w}_{(t-1)}, \tilde{z}^{h/w}) = a^{h/w} \cdot \text{sigm}(\text{sigm}^{-1}(o^{h/w}_{(t-1)}) + \tilde{z}^{h/w})$$

$$f^{\text{depth}}(o^{\text{depth}}_{(t-1)}, \tilde{z}^{\text{depth}}) = \text{sigm}(\text{sigm}^{-1}(o^{\text{depth}}_{(t-1)}) + \tilde{z}^{\text{depth}})$$

Here sigm stands for the sigmoid function. Finally, f^{where} is a complicated function (imported from SQAIR) that is difficult to summarize succinctly here; see the accompanying code for details. Investigating whether the *where* update can be replaced with something simpler should be a target of future ablation studies.

Recurrent hidden state. For propagation we assume an additional object attribute $o^{\text{hidden}}_{(t)}$, which stores the hidden state of a recurrent neural network p^{rnn}_{ϕ} and is fully deterministic. This hidden state of each object is provided as an additional argument to the spatial attention module which builds features for propagation. After each propagation step, the hidden state for each object is updated independently by running the recurrent network, taking the new object attributes as input:

$$\tilde{o}^{\text{hidden}} = p^{\text{rnn}}_{\phi}(o^{\text{hidden}}_{(t-1)}, \tilde{o}^{\text{where}}, \tilde{o}^{\text{what}}, \tilde{o}^{\text{depth}}, \tilde{o}^{\text{pres}})$$

We can think of the hidden state as providing the propagation module with a deterministic path from an object's past to the present. Newly discovered objects are given a default initial hidden state. The value of the default hidden state is a trainable parameter.

A.2 Rendering

The rendering module is the sole constituent of the VAE decoder. It takes in the current set of objects $o_{(t)}$ and renders them into a frame. We start by focusing on a single object with index k, and drop temporal indices. First, an appearance map and a partial transparancy map are predicted:

$$\beta^{\text{logit}}_k, \xi^{\text{logit}}_k = r^{\text{obj}}_{\theta}(o^{\text{what}}_k)$$

$$\beta_k = \text{sigmoid}(\mu^{\beta} + \sigma^{\beta}\beta^{\text{logit}}_k)$$

$$\xi_k = \text{sigmoid}(\mu^{\xi} + \sigma^{\xi}\xi^{\text{logit}}_k)$$

The appearance map β_k has shape $(H_{\text{obj}}, W_{\text{obj}}, 3)$, while the partial transparency map ξ_k has shape $(H_{\text{obj}}, W_{\text{obj}}, 1)$, for integers $H_{\text{obj}}, W_{\text{obj}}$. Meanwhile $\sigma^{\beta}, \mu^{\beta}, \sigma^{\xi}$ and μ^{ξ} are scalar hyperparameters that can be used to control the relative speed with which appearance and transparency are trained.

ξ_k is multiplied by o^{pres}_k to ensure that objects are only rendered to the image to the extent that they are present, yielding a final transparency map:

$$\alpha_k = \xi_k \cdot o^{\text{pres}}_k$$

Next we combine α_k with o^{depth}_k to get an *importance* map:

$$\gamma_k = \alpha_k \cdot o^{\text{depth}}_k$$

For each object, an inverse spatial transformer parameterized by o^{where}_k is then used to create image-sized versions of these three maps, with the input maps placed in the correct location:

$$\alpha'_k, \beta'_k, \gamma'_k = \tau^{-1}([\alpha_k, \beta_k, \gamma_k], o^{\text{where}}_k)$$

For each location in one of these image-sized maps, the value is obtained from a corresponding location in the input space, dictated by the location parameters o^{where}_k. However, some of these locations will lie "outside" of the input map, and for these locations we use a default value. In particular, we use a default of 0 for α and β, and $-\infty$ for γ.

To obtain the output frame, the image-sized appearance maps are combined by weighted summation. For each pixel we take the softmax (over objects) of the importance values, and weight each object by the resulting value. We also weight by α' to implement transparency. Thus we have:

$$\hat{x}_{(t)} = \frac{\sum_{k=0}^{K-1} \beta'_k \alpha'_k e^{\gamma'_k/\lambda}}{\sum_{\ell=0}^{K-1} e^{\gamma'_\ell/\lambda}}$$

Here λ is a hyperparameter that acts as the temperature of the softmax. When a given pixel is within the bounding boxes of two or more objects, the softmax implements a differentiable approximation of relative depth, and objects with larger values for o^{depth} (and thus larger values for γ, all else being equal) are rendered on top of objects with lower values. The default of $-\infty$ used when spatially transforming γ ensures that objects do not contribute to the softmax for pixels that are not within their bounding box.

Note that all computations in this section can be parallelized to a high degree (i.e. across objects). However, for

large frames this scheme can still be expensive in terms of both memory and computation. Thus in practice we use an equivalent (but more complex) implementation that avoids explicitly constructing image-sized maps for each object. The output of rendering is an image with dimensions $(H_{\text{inp}}, W_{\text{inp}}, 3)$; to obtain $g_\phi(x|z)$, we use this image to parameterize a set of (conditionally) independent Bernoulli random variables, one for each pixel and channel.

B Spatial Attention

In this section we provide details on the spatial attention steps used in the discovery and propagation modules. Both are similar to Neural Physics Engine (Chang et al. 2016), though more so for the Propagation version.

B.1 Spatial Attention for Discovery

Recall that in the discovery module, spatial attention is used to obtain, for each discovery unit, a feature vector summarizing nearby propagated objects. The main motivation is to allow the discovery module to avoid rediscovering objects that are already accounted for:

$$v^{\text{td}}_{(t)} = \text{SpatialAttention}^{\text{disc}}_\phi(\tilde{o}_{(t)}, \sigma)$$

We first narrow our focus to a single discovery unit with indices ij. For every propagated object $\tilde{o}_k$ (dropping temporal indices here), we first use an MLP d^{spatial}_ϕ to compute a feature vector that is specific to ij. As input to this MLP, we supply the object $\tilde{o}_k$, except that we replace the position attributes $\tilde{o}^y_k$, $\tilde{o}^x_k$ with *relative* position attributes $\tilde{o}^{y'}_{ij,k}$ and $\tilde{o}^{x'}_{ij,k}$. Here we are assuming that what is important is the position of the propagated objects relative to the grid cell, rather than their absolute positions. Noting that the center of grid cell ij has location

$$((i + 0.5) \cdot c_h, (j + 0.5) \cdot c_w)$$

we have

$$\tilde{o}^{y'}_{ij,k} = \tilde{o}^y_k - (i + 0.5) \cdot c_h$$
$$\tilde{o}^{x'}_{ij,k} = \tilde{o}^x_k - (j + 0.5) \cdot c_w$$

The spatial attention module then just sums these feature vectors over propagated objects k, weighted by a Gaussian kernel:

$$v^{\text{td}}_{ij} = \sum_{k=0}^{K-1} G(\tilde{o}^{y'}_{ij,k}, \tilde{o}^{x'}_{ij,k}, \sigma) \cdot d^{\text{spatial}}_\phi(\tilde{o}^{\setminus yx}_k, \tilde{o}^{y'}_{ij,k}, \tilde{o}^{x'}_{ij,k})$$

where G is the density of a 2 dimensional Gaussian, and $\tilde{o}^{\setminus yx}_k$ contains all attributes of $\tilde{o}_k$ except y and x. Note that this can be computed for all ij and k in parallel (except for the summation over k).

B.2 Spatial Attention for Propagation

In the propagation module, spatial attention is used to compute a feature vector for each object from the previous step, which is subsequently used to predict updates to the attributes of the object.

$$u^{\text{td}}_{(t)} = \text{SpatialAttention}^{\text{prop}}_\phi(o_{(t-1)}, \sigma)$$

This vector is supposed to take into account attributes of the object itself, as well as attributes of nearby objects. This should allow the updates to take into account the effect of nearby objects on the target object.

We first narrow our focus to a target object with index ℓ. We use an MLP p^{td}_ϕ to compute an initial feature vector:

$$u^{\text{td}'}_\ell = p^{\text{td}}_\phi(o_\ell)$$

Then for every object k we compute the position of object k relative to object ℓ:

$$o^{y'}_{\ell,k} = o^y_k - o^y_\ell$$
$$o^{x'}_{\ell,k} = o^x_k - o^x_\ell$$

Next we use an MLP p^{spatial}_ϕ to get feature vectors for o_k in the context of target object o_ℓ. This is similar to discovery, except the MLP also takes $u^{\text{td}'}_\ell$ as an argument. The results are summed and weighted by a Gaussian kernel, and then $u^{\text{td}'}_\ell$ is added in:

$$u^{\text{td}}_\ell = u^{\text{td}'}_\ell + \sum_{k=0}^{K-1} G(o^{y'}_{\ell,k}, o^{x'}_{\ell,k}, \sigma) \cdot p^{\text{spatial}}_\phi(o^{\setminus yx}_k, o^{y'}_{\ell,k}, o^{x'}_{\ell,k}, u^{\text{td}'}_\ell)$$

We can think of the second term as computing the additive effect of nearby objects on the target object. Again this can be computed for all ℓ and k in parallel (except the summation).

C Baseline Algorithm: ConnComp

We compare against a simple baseline algorithm called ConnComp (Connected Components), which works as follows. For each frame a graph is created wherein the pixels are nodes, and two pixels are connected by an edge if and only if they are adjacent and have the same color. We then extract connected components from this graph, and call each connected component an object. This yields a set of objects for each frame. In order to track objects over time (i.e. to assign persistent identifiers to the objects), we employ the Hungarian algorithm to find matches between detected objects in each pair of successive frames (Kuhn 1955). As matching cost we use the distance between object centroids, and require that matching objects have the same color (objects pairs with mismatched colors are assigned a cost of ∞). The performance of ConnComp can be interpreted as a measure of the difficulty of the dataset; it will be successful only to the extent that objects can be tracked by color alone.

D Experiment Details

D.1 Scattered MNIST

Each video had spatial size 48×48 pixels. MNIST digits were resized to 14×14 pixels (their original size being 28×28. Initial digit velocities vectors were sampled uniformly, with a fixed magnitude of 2 pixels per frame. Digits passed through one another without event, and bounce of the edges of the frame. Initial digit positions were sampled randomly, one digit at a time. If the cumulative overlap between a newly sampled digit position and existing digit positions exceeded

a threshold (98 pixels), the digit position was resampled until the threshold was not exceeded. This allowed us to control the amount of overlap between digits on the first frame. On subsequent frames, no overlap limit is enforced, and indeed because objects pass through one another and bounce of frame borders, the degree of overlap can be quite high in subsequent frames. In such cases, networks need to use information about the locations and trajectories of the overlapping objects at previous timesteps in order to track well.

The 60,000 MNIST digits were divided up into 80% training set, 10% validation set, and 10% test set. Videos for training set were created by uniformly sampling the number of digits to put in the video (from distribution Uniform$(1, 6)$ or Uniform$(1, 12)$ depending on the training condition), and then sampling that number of digits from the set of training digits. Similar approaches were taken for validation and testing; this ensures that at test time, networks are seeing digits that they have never seen before. We created 60,000 training videos and 1000 validation videos for each training condition, and 1000 videos for each testing condition (i.e. for each position on the x-axis of the plots in Figure 5 of the main text).

D.2 Scattered Shapes

Videos in the Scattered Shapes dataset were created in a manner similar to Scattered MNIST, with a few exceptions. Possible colors were red, green, blue, cyan, yellow, magenta, and possible shapes were circle, diamond, star, cross, x. All shape/color combinations were present in training, validation and test datasets. Initial pixel velocities had a fixed magnitude of 5 pixels per frame rather than 2. Additionally, when adding a shape to a video, it's size was randomized by choosing each spatial dimension from a Normal$(\mu = 14, \sigma = 1.4)$ distribution.

E Model and Training Details

E.1 SILOT

SILOT was implemented in tensorflow, and is available online at `https://github.com/e2crawfo/silot`. Hyperparameters are listed in Table 1. Recall that we trained SILOT using a curriculum, starting training on the initial 2 frames of each video, and increasing by 2 frames every N_{curric} timesteps until the network is training on full videos. Once the network is training on full frames, we begin an early stopping regime wherein we train until the measure of performance (in most cases, MOTA on the validation set) does not improve for 30,000 training steps (i.e. using a "patience" value of 30,000, at the level of training steps rather than epochs), triggering an early stop. Each time an early stop is triggered, we go back to the set of weights that has the best performance so far, divide the learning rate by 3, and resume training. Training ends once early stopping has been triggered 3 times. We take as our final hypothesis the set of weights that achieved the best performance on the validation set.

Finally, for the first 1000 update steps we do not backpropagate gradients through *where*, *depth*, or *pres* attributes. Without this initial period, the component networks responsible for encoding and decoding appearance information (particularly d_ϕ^{obj}, p_ϕ^{obj} and r_θ^{obj}) are initially poor at reconstructing any part of the scene, and the network often realizes that it can reduce the reconstruction loss by simply turning all of the objects off or shrinking them into oblivion. Both of these possibilities are very poor local minima, and need to be avoided. Additionally, we use a prior for the *pres* attributes which starts by encouraging the network to use many objects, but over time is annealed into a prior that encourages using few objects (using a very similar approach to SPAIR (Crawford and Pineau 2019)).

SILOT contains a relatively large number of component networks. Table 2 lists these networks, along with descriptions of their role and architecture. The architecture of the backbone convolutional network d_ϕ^{bu} is:

```
[Conv(n=128, f=4, s=3, nl=RELU)
 Conv(n=128, f=4, s=2, nl=RELU),
 Conv(n=128, f=4, s=2, nl=RELU),
 Conv(n=128, f=1, s=1, nl=RELU),
 Conv(n=128, f=1, s=1, nl=RELU),
 Conv(n=128, f=1, s=1, nl=None)]
```

where n gives the number of filters, f gives the filter width, s gives the stride and nl gives the non-linearity used. No pooling is used at any point.

E.2 SQAIR

For our implementation of SQAIR, we used a lightly modified version of the original implementation: `https://github.com/akosiorek/sqair`.
Initial tuning of hyperparameters was performed by hand (starting from the default values) until a reasonable range of values for the Scattered MNIST dataset was identified. As model-selection criteria, we used the MOTA (a measure of object tracking performance) averaged over 1–6 digits for the 1–6 training case, and averaged of 1–12 digits for the 1–12 training case. We ran an additional grid search over select hyperparameters.

F Experiment Visualizations

A visualization of a forward pass of SILOT on the Scattered MNIST task is shown in Figure F1, and on the Scattered Shapes task in Figure F2.

G Additional Experiments

G.1 Scattered MNIST - Propagating with Prior

As detailed in Section 3.8, we trained a learned prior for the propagation latent variables in addition to the static prior. This can be viewed as a duplicate of the main propagation module, except that it does not have access to the frame each time step. Here we test the performance of that module in the Scattered MNIST task. The evaluation procedure is similar to the one used in SQAIR for the same purpose (Kosiorek et al. 2018), and runs as follows. For the first 3 timesteps, the regular network is used (with the regular propagation module), in order to discover objects and estimate their initial trajectory. For the remaining 5 frames the discovery module is

Description	Variable	Value
Initial learning rate		0.0001
Batch size		16
Max gradient norm		10.0
Optimizer		Adam
Patience		30,000
# of training steps for each stage of curriculum	N_{curric}	40,000
Probability of discovery dropout	p_{dd}	0.5
Number of propagated/selected objects	K	16
Dimension of *what* attribute	A	64
Anchor box size in pixels	(a_h, a_w)	(48, 48)
Grid cell size in pixels (determined by structure of d_ϕ^{bu})	(c_h, c_w)	(12, 12)
Bounds on distance between object center and grid cell center in disc.	(b^{min}, b^{max})	(-0.5, 1.5)
Standard dev. of Gaussian kernel for spatial attention	σ	0.1
Prior on $\bar{z}^h$, $\bar{z}^w$		Normal$(\mu = -2.2, \sigma = 0.5)$
Prior on $\bar{z}^y$, $\bar{z}^x$		Normal$(\mu = 0, \sigma = 1)$
Prior on $\bar{z}^{\text{what}}$		Normal$(\mu = 0, \sigma = 1)$
Prior on $\bar{z}^{\text{depth}}$		Normal$(\mu = 0, \sigma = 1)$
Prior on $\bar{z}^{\text{pres}}$		See Section E.1
Prior on $\tilde{z}^h$, $\tilde{z}^w$		Normal$(\mu = 0, \sigma = 0.3)$
Prior on $\tilde{z}^y$, $\tilde{z}^x$		Normal$(\mu = 0, \sigma = 0.3)$
Prior on $\tilde{z}^{\text{what}}$		Normal$(\mu = 0, \sigma = 0.4)$
Prior on $\tilde{z}^{\text{depth}}$		Normal$(\mu = 0, \sigma = 1)$
Prior on $\tilde{z}^{\text{pres}}$		See Section E.1
Appearance offset and scale	μ^β, σ^β	(0.0, 2.0)
Transparency offset and scale	μ^ξ, σ^ξ	(5.0, 0.1)
Rendering softmax temperature	λ	0.25
Rendered object size	(H_{obj}, W_{obj})	(14, 14)

Table 1: Base hyperparameter values for SILOT.

Figure F1: Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered MNIST task containing 8 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have passed completely through other objects (e.g. 5 with the green box, 3 with the grey box).

	Description	Architecture
d_ϕ^{bu}	Computes bottom-up features for disc. grid cells	See Section E.1
d_ϕ^{spatial}	Computes features of propped objects in disc. attention	FC([64, 64], RELU)†
d_ϕ^{fuse}	Combines top-down and bottom-up info in disc	FC([100, 100], RELU)†
d_ϕ^{where}	Predicts params. for posterior dist. over $\bar{z}^{\text{where}}$	FC([100, 100], RELU)†
d_ϕ^{obj}	Processes glimpse in disc.	FC([256, 128], RELU)†
d_ϕ^{what}	Predicts params. for posterior dist. over $\bar{z}^{\text{what}}$	FC([100, 100], RELU)†
d_ϕ^{depth}	Predicts params. for posterior dist. over $\bar{z}^{\text{depth}}$	FC([100, 100], RELU)†
d_ϕ^{pres}	Predicts params. for posterior dist. over $\bar{z}^{\text{pres}}$	FC([100, 100], RELU)†
p_ϕ^{td}	Computes object features in prop. attention	FC([64, 64], RELU)
p_ϕ^{spatial}	Computes object-pair features in prop. attention	FC([64, 64], RELU)
p_ϕ^{glimpse}	Predicts params for initial prop. glimpse	FC([100, 100], RELU)
p_ϕ^{bu}	Processes initial prop. glimpse	FC([256, 128], RELU)
p_ϕ^{where}	Predicts params. for posterior dist. over $\tilde{z}^{\text{where}}$	FC([100, 100], RELU)
p_ϕ^{obj}	Processes second glimpse in prop.	FC([256, 128], RELU)
p_ϕ^{what}	Predicts params. for posterior dist. over $\tilde{z}^{\text{what}}$	FC([100, 100], RELU)
p_ϕ^{depth}	Predicts params. for posterior dist. over $\tilde{z}^{\text{depth}}$	FC([100, 100], RELU)
p_ϕ^{pres}	Predicts params. for posterior dist. over $\tilde{z}^{\text{pres}}$	FC([100, 100], RELU)
p_ϕ^{rnn}	Updates deterministic hidden state	GRU(128)
r_θ^{obj}	Predicts object appearances in rendering.	FC([128, 256], RELU)

Table 2: Component neural networks in SILOT. FC([N, N], RELU) is a sequence of 3 fully-connected layers (2 hidden layers each with N units, one output layer). The RELU non-linearity is applied only at the hidden layers, the output is left unconstrained. †Recall that in Section 3.3 (Discovery) we said that we use convolutional networks in Discovery; however, since those networks generally use a stride and filter size of 1, they are equivalent to applying a single fully-connected layer independently to each spatial location. Thus, here we are listing the fully-connected equivalent of the convolutional networks that were actually used.

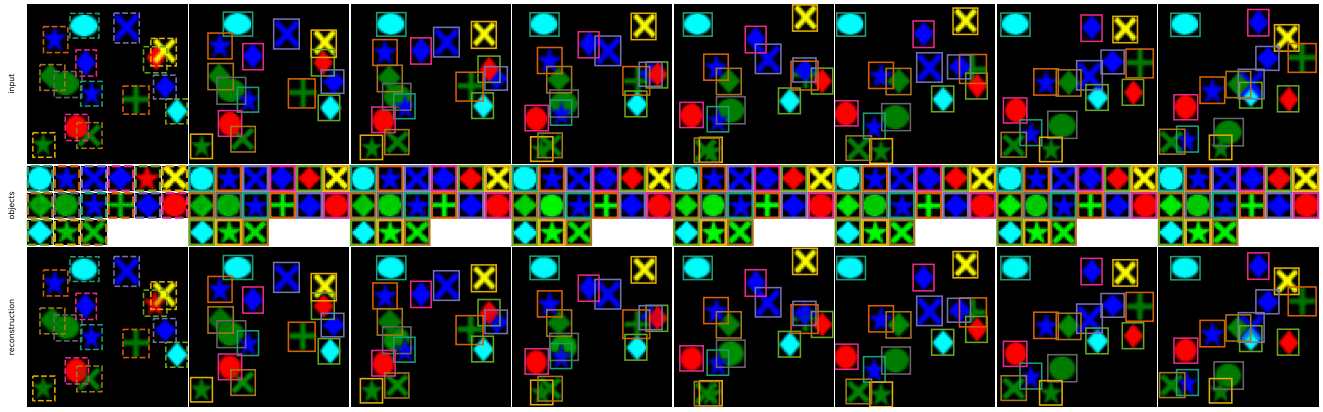

Figure F2: Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered Shapes task containing 15 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have been heavily occluded by other objects (e.g. green cross with the orange box that starts near the center).

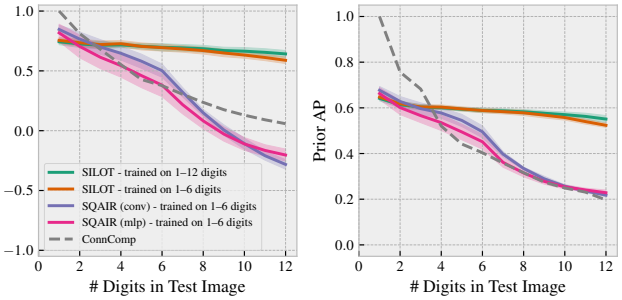

Figure G3: Probing ability of the learned prior propagation modules to predict object trajectories without access to the input frames.

deactivated, and the prior propagation module is used instead of the main propagation module. We can do this for both SILOT and SQAIR.

Evaluation metrics were computed only on the final 5 frames. We are thus testing the ability of the prior propagation module to predict the trajectories of the objects detected in the first 3 frames by the main network. Results are shown in Figure G3.

References

[Chang et al. 2016] Chang, M. B.; Ullman, T.; Torralba, A.; and Tenenbaum, J. B. 2016. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*.

[Crawford and Pineau 2019] Crawford, E., and Pineau, J. 2019. Spatially invariant, unsupervised object detection with convolutional neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*.

[Kosiorek et al. 2018] Kosiorek, A.; Kim, H.; Teh, Y. W.; and Posner, I. 2018. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Advances in Neural Information Processing Systems*, 8606–8616.

[Kuhn 1955] Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2):83–97.