A Robust Online Multi Object Tracking Using Bipartite Graph

Jaime Eduardo Fajardo Muñoz

Institute of Aerospace Studies, University of Toronto
jaime.fajardo@mail.utoronto.ca

Abstract

This paper presents the development of a multi-object tracking (MOT) system, integrating the Faster R-CNN detector with a custom-build bipartite graph network. Utilizing the Hungarian algorithm for object matching, the system aims to address the complexities of tracking multiple objects in dynamic environments. My computational less demanding approach achieved a Multiple Object Tracking Accuracy (MOTA) of 58.8 and an ID F1 Score (IDF1) of 56.6. Future enhancements include the adoption of an offline tracking approach for better information utilization, the integration of a Kalman filter for improved prediction of bounding box positions, and the exploration of more advanced detection methods, such as YOLO, to further enhance performance. The step by step code will be available at https://github.com/jefajardom/MOT-Using-Bipartite-Graph

1 Introduction

Multi object tracking (MOT) is a fundamental problem and longstanding goal in computer vision [1] that involves tracking multiple objects in a video sequence over time. The goal is to estimate the trajectory of object of interest in a video frame and associate the correct object with its corresponding trajectory. It has many application, including surveillance [2], traffic monitoring [3], and robotics [4]

MOT presents challenging problems due to factors such as occlusion, appearance changes, and motion blur [5]. Traditionally, these challenges have been addressed through a *tracking by detection* paradigm [5]. This approach involves applying a detector to each video frame to propose object locations, followed by a data association process to connect these detections over time, creating object trajectories.

Recent trends, however, indicate a shift in methodology. Based on insights derived from a benchmark conducted during a MOT challenge competition involving hundreds of trackers, as detailed in [6], it has been observed that while the conventional data association method maintains good accuracy, a more precise approach known as *tracking by regression* is emerging. This method effectively merges the tasks of tracking and detection, offering computational efficiency and real-time tracking capabilities. Furthermore, graph-based techniques have gained attention for their ability to model data associations effectively [7], framing MOT as an assignment problem where nodes represent object detections across frames, and edges denote the associations between these detections.

Therefore, in this paper, I propose a less computational expensive MOT system using a bipartite graph, following the pipeline from [7]. In Section 2, I will discuss existing methods and technologies in multi-object tracking, identifying gaps or limitations in current approaches. Then, in Section 3, I will present an overview of the bipartite graph approach, including its integration, construction, and training methods. Finally, in Section 4, I will evaluate this method on the MOT16 benchmarks [8].

2 Literature Review

2.1 Related Work

Multi-object tracking (MOT) has seen a variety of approaches and methodologies, each attempting to address the inherent challenges of the field.

Deep Learning Approaches: With the advent of deep learning, MOT has witnessed significant improvements. Convolutional Neural Networks (CNNs) have been extensively used for feature extraction in tracking scenarios. The emergence of R-CNN [9] and its variants, like Faster R-CNN[10], have revolutionized object detection, a critical component of MOT.

Tracking-by-detection methods create object trajectories over time using graph-based optimization techniques, which, despite being effective, can be computationally intensive for real-time applications [7]. Appearance-based tracking methods utilize advanced image recognition to track objects, but they struggle in crowded settings with multiple occlusions[11]. Motion modeling, another technique, predicts future object locations, though accurately converting complex 3D motions to 2D remains challeng-

ing[12].

Tracking-by-regression predicts new object locations without linking detections across frames. While effective, it often requires supplementary re-identification [13] and motion models, as well as graph techniques, to enhance accuracy and maintain object identities[14].

Online Tracking Methods : Online tracking algorithms process video frames sequentially and update the tracks with each new frame[15]. A popular method in this category is the Kalman Filter [16], which is widely used for its simplicity and effectiveness in predicting object positions. Another approach is the use of particle filters, which, despite being computationally intensive, offer greater flexibility in handling non-linear motions.

Offline Tracking Methods: These methods, also known as batch processing methods, process the entire video data before tracking [7]. While they offer the advantage of utilizing future frame information, they are not suitable for real-time applications.

Data Association Techniques: Data association, a crucial step in MOT, has been addressed using various methods. Hungarian algorithm-based approaches are common for frame-to-frame association. Another significant technique is the use of bipartite graphs and linear programming to solve association problems [13]

2.2 Gap Analysis

While there have been numerous advancements in MOT, several gaps and limitations persist in current approaches:

Real-Time Processing: Many state-of-the-art MOT systems struggle to balance accuracy with real-time processing capabilities. This is particularly challenging in scenarios with high-density scenes and rapid object movements.

Occlusions and Identity Switches: Handling occlusions and preventing identity switches remain major challenges. Existing methods often lose track of objects during prolonged occlusions or in crowded scenes, leading to reduced tracking accuracy.

Integration of Detection and Tracking: Although there have been significant improvements in object detection algorithms, their integration with tracking systems is not always optimal. This results in either increased false positives or missed detections, impacting the overall efficacy of the MOT system.

Resource Efficiency: Many advanced MOT methods require substantial computational resources, limiting their applicability in resource-constrained environments.

Adaptability: The ability of MOT systems to adapt to different scenarios, such as varying lighting conditions, camera angles, and object densities, is often limited.

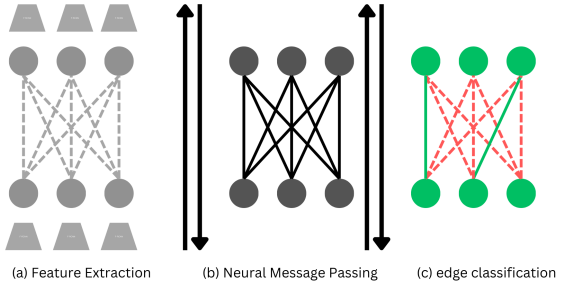

Figure 1: *Overview of the Bipartite Graph Network. After obtaining the detection from the Faster-RCNN as inputs for the network, (a) A graph is created where each detection is represented as a node. Nodes from different frames are interconnected through edges. (b) Subsequently, these embeddings undergo propagation within the graph over a set number of iterations, employing neural message passing. (c) At the end of this process, the refined embeddings from the neural message passing are utilized to categorize the edges as either active (indicated in green) or inactive (indicated in red). In the training phase, the cross-entropy loss of our predictions is calculated relative to the ground truth labels. Finally, based on our classification scores, final trajectories were obtained.*

The method proposed in this paper aims to address some gaps by developing a robust online MOT system that effectively integrates advanced object detection techniques with a bipartite graph-based tracking approach. The focus is on enhancing real-time processing capabilities while maintaining high accuracy and minimize the training time process.

3 Methodologies

3.1 Graph Construction

A bipartite graph was chosen for its simpler structure, where vertices can be divided into two disjoint sets with edges connecting vertices from opposite sets. This approach provides a baseline for potentially incorporating more complex layers into the graph network in the future. Following the pipeline from [7], this method was adapted to suit the bipartite graph structure as seen in Figure 1

As an overview of the network flow, in the tracking problem, it was started with a collection of object detections $O = \{o_1, ..., o_n\}$ encompassing all objects across the video frames, with n representing the total count of these objects. Each object detection, denoted by $o_i = (a_i, p_i, t_i)$. Here, a_i represents the bounding box's raw pixel data, p_i specifies its 2D image coordinates, and t_i indicates the time at which the detection occurred. A trajectory is formed by a sequence of

such detections, ordered by time, and is represented as $T_i = \{o_i, ..., o_{i_{n_i}}\}$ with n_i being the count of detections comprising the i trajectory. The objective of Multi-Object Tracking (MOT) is to determine a set of trajectories $T_* = \{T_1, ..., T_m\}$ that most accurately correspond to the observed detections O.

To tackle this, it was conceptualized the problem using an undirected graph $G = (V, E)$, where $V := \{1, ..., n\}$, $E \subset V \times V$ Each node $i \in V$ corresponds to a unique detection $o_i \in O$. The edges E are arranged such that every pair of detections across different frames is interconnected, facilitating the identification of trajectories, even when some detections are missed. This setup transforms the task of segregating the original set of detections into distinct trajectories into a problem of grouping nodes in the graph into separate clusters. Consequently, each trajectory T_i in the scene is equivalent to a cluster of nodes $\{i_1, ..., i_{n_i}\}$ in the graph, and vice versa.

Now, in order to make sure that each edge just have one active edge or trajectory (one edge set to 1 and rest to 0), it was followed partially the classical mini-cost flow view of MOT [17] in which define a variable $y_{(i,j)}$ for each timestamps (i, j) and associate a cost $c_{(i,j)}$.The ultimate partition is determined through an optimization process:

$$
\begin{aligned}
&min_y \sum_{(i,j)\in E} c_{(i,j)} y_{(i,j)} \\
&subject\ to: \sum_{(j,i)\in E\ s.t\ t_i > t_j} y_{(j,i)} \leq 1 \\
&\sum_{(i,k)\in E\ s.t\ t_i > t_k} y_{(i,k)} \leq 1 \\
&y_{(i,j)} \in \{0, 1\}, (i, j) \in E
\end{aligned}
$$

However, [7] suggests a direct method for predicting the active edges in the graph, essentially predicting the eventual state of the binary variable y. It was followed this approach this by framing the problem as an edge classification task, with the binary variables y serving as classification labels. In essence, it was utilized the traditional network flow model that it was previously described, transforming the MOT challenge into a task that is entirely learnable, of course, this will be further explained in the next steps.

3.2 Training the Regression Model Detector

To initialize the bipartite graph, the next step involved employing a regression-based detector, specifically Faster R-CNN [10] with a ResNet50 feature extractor. This model was trained using the native PyTorch implementation of Faster R-CNN, available at PyTorch Tutorials [1]. The precomputed features from this model serve as the foundation for object detection in our system.

3.3 Feature Encoding

After extracting feature embeddings from the Faster R-CNN, these features were applied to each node representing a bounding box. Nodes were then interconnected through edges. For each pair of detections (edges), a feature vector was computed, encapsulating their relative positional attributes. An edge is deemed *active* if, after solving the graph model, it indicates that the connected bounding boxes belong to the same object.

To develop a representation that captures the relative positioning, size, and time differences for every pair of detections across distinct frames. When considering a pair of detections o_i and o_j, each having different timestamps $t_i \neq t_j$, it is possible to analyze their bounding boxes. These boxes are defined by their top-left corner coordinates, height, and width, denoted as (x_i, y_i, h_i, w_i) and (x_j, y_j, h_j, w_j), respectively. it was calculated their relative size and distance using these parameters[7]:

$$
\left(\frac{2(x_j - x_i)}{h_i + h_j}, \frac{2(y_j - y_i)}{h_i + h_j}, log\frac{h_i}{h_j}, log\frac{w_i}{w_j} \right) \tag{1}
$$

This calculation results in a feature vector based on coordinates. Then, it was merged this vector with the time interval $t_j - t_i$ and the relative appearance characteristics. This combined data is then processed feed to the bipartite graph nodes to to generate the initial edge embedding.

3.4 Neural Message Passing

Implementing Neural Message Passing (NMP) as described in [7] involved performing a series of message-passing steps over the graph. This process refines the features within the nodes, essentially facilitating communication between them. NMP can be intuitively understood as each node assessing the features of its neighboring node to determine if they represent the same object.

Recalling subsection 3.1, considering a graph G represented as $G = (V, E)$. For each node $i \in V$, let there be an initial node embedding $h_i^(0)$, and for each edge $(i, j) \in E$ an initial edge embedding $h(0)(i, j)$. The primary objective of the network is to devise a mechanism that facilitates the propagation

[1] https://pytorch.org/tutorials/intermediate/torchvision_tutorial

of information present in the feature vectors of nodes and edges throughout the graph G.

This propagation process is structured in the form of embedding updates for both nodes and edges, commonly referred to as *message passing steps* as detailed in [18]. The methodologies break down each message passing step further into two distinct update phases: an update from nodes to edges denoted as $(v \rightarrow e)$ and an update from edges to nodes $(e \rightarrow v)$. These updates are executed in a sequential manner over a predetermined number of iterations, represented as L. For each $l \in \{1, ..., L\}$, the updates follow a general format as presented in [7]:

$$(v \rightarrow e) \quad h_{(i,j)}{}^{(l)} = N_e([h_i{}^{l-1}, h_j{}^{l-1}, h_{(i,j}{}^{l-1}]) \tag{2}$$

$$(e \rightarrow v) \quad m_{(i,j)}{}^{(l)} = N_v([h_i{}^{l-1}, h_{i,j}{}^{l}]) \tag{3}$$

$$h_i{}^{(l)} = \Phi(\{m_{(i,j)}{}^{(l)}\}_{j \in N_i}) \tag{4}$$

In this context, N_e and N_v symbolize trainable functions, and were initialized by the faster R-CNN. Φ represents an operation that is invariant to order, for instance, a summation, maximum, or average. It is important to note that after L iterations, each node in the graph will have assimilated information from all other nodes that are located at a distance of L within the graph. Implement this mythology in this bipartite graph was less less complex than [7], since the graph has two set of nodes that not overlap.

3.5 Matching

After obtaining a prediction, it is necessary to match it with the detection from the current frame. So, the Hungarian algorithm[2] was employed to solve the assignment problem. This algorithm optimally matches newly detected objects to past tracks, minimizing the overall cost, which is based on re-identification (ReID) distance between frames.

To optimize the matching process, several strategies were implemented[3]. Firstly, tracks that cannot be matched with past detections are killed, and new trajectories are initiated for objects that remain unidentified. Additionally, an attribute system was introduced for the tracker, categorizing tracks as *active* or *inactive*. A track is marked *active* if a match is found in the current frame, while those without a match are labeled *inactive*, which could be due to occlusions. Moreover, the system incorporates a *patience* mechanism. If a track remains inactive for more than a predetermined number of frames, it is then terminated

[2] https://www.hungarianalgorithm.com/hungarianalgorithm.php

[3] https://dvl.in.tum.de/slides/cv3dst-ws19/5.MOT2.pdf

Figure 2: *Tracking result for MOT16-9 frame 20*

3.6 Training

Finally, the resulting edge embeddings were used for a binary classification task to determine active versus non-active edges.

Given the nature of this output, a Cross-Entropy loss function was utilized to train the model predictions.

4 Results

The method was trained and tested with the MOT16 challenge data[8]. The method was trained and test it with the MOT16 challenge data[8]. This provides 7 train and 7 test video sequences with multiple objects (pedestrians) per frame. It includes many challenging scenarios with camera movement, high crowdedness and object occlusions. For training, the 7 train video sequences were divided as follows:

- 5 set for training
- 2 set for validation
- 7 set for testing

After training and testing our model, it was possible to to achieve tracking frames as illustrated in Figure 2

Of course, it was necessary to determine a metric of accuracy to evaluate the performance of the tracker effectively.

4.1 Metrics

Perhaps the most widely used metric to evaluate a tracker's performance [1][7][19] is Multiple Object Tracking Accuracy (MOTA) that focus on object coverage and F1 Score (IDF1) that measures the identity preservation.

MOTA combines three different errors as follows:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \tag{5}$$

Table 1: *Report result from data sets MOT16-2 and MOT16-11*

Sequence	MOTA	IDF1	FP	FN
MOT16-2	49.6	48.6	390	8873
MOT16-11	77	70	266	1871
Overall	58.8	56.6	656	10744

Table 2: *Comparison of modern multi-object tracking methods evaluated on MOT16 test sets for different public detection processing*

Method	MOTA	IDF1	FP	FN
STRSORT	78	78.3	12981	25660
OUTrack	69.3	67.5	10647	44059
Mine	58.8	56.6	656	10744
Tracktor	56.2	54.9	2394	76844

where t is the frame index, GT is the number of ground truth objects, FP is the false positive bounding boxes not corresponding to any ground truth, FN is the false negative ground truth boxes not covered by any bounding box and $IDSW_t$ is the bounding boxes switching the corresponding ground truth identity.

Fortunately, the MOT competition has made available a library [4], that implement the evaluation in an *offline mode*. The py-motmetrics library offers a Python-based solution for benchmarking multiple object trackers (MOT), including metrics such as MOT and IDF1 that was implemented in the main code and obtained the results in Table 1.

The overall result for MOTA for the presented method was 58.8% and IDF1 of 56.6%.

Then, the results were compared with top state of the art trackers from the competition in Table 2. Also, it was included in this table one of the best trackers in 2019 [13] for reference and discussion in the following section.

5 Discussion

The development of a multi-object tracking (MOT) system presented in this paper relies on the Faster R-CNN detector and a custom-built bipartite graph network, employing the Hungarian algorithm for effective matching.

5.1 Comparative Analysis

Performance Metrics: The system achieved a MOTA (Multiple Object Tracking Accuracy) of 58.8 and an IDF1 (ID F1 Score) of 56.6. While these results fall short of the current best tracker, which boasts a MOTA of 78 and an IDF1 of 78.3, they still represent a noteworthy achievement. **Benchmarking Against Previous Trackers**: Importantly, the system outperforms one of the leading trackers from 2019, which had a MOTA of 56.2 and an IDF1 of 54.9. This comparison highlights the efficacy of the developed system, particularly in terms of tracking accuracy and identity preservation.

5.2 Computational Considerations

Resource Intensity: It is important to emphasize that, compared to state-of-the-art trackers, the training of the bipartite neural network is computationally less demanding. This efficiency stems from the simpler architecture and fewer connections of the bipartite graph network. However, this streamlined design may impact the system's performance in resource-intensive scenarios or real-time applications, potentially limiting its applicability in such environments.

5.3 Conclusion and Future Work

In conclusion, the developed MOT system demonstrates promising decent capabilities in tracking multiple objects, balancing detection accuracy with identity tracking. The open source code shared in this paper can works as a step by step tutorial for creating a Bipartite Neural Network for MOT from scratch. However, future work is necessary and could focus on the following areas:

1. **Offline Tracking Approach**: Implementing an offline tracking method could potentially enhance the system's performance. Offline tracking would allow the system to utilize future frame information, thereby improving tracking accuracy, especially in complex scenarios.
2. **Integration of Kalman Filter**: Incorporating a Kalman filter to predict the position of bounding boxes in subsequent frames could significantly improve the tracking consistency. This would be particularly useful in handling occlusions and rapid object movements.
3. **Adoption of a Superior Detector** : Experimenting with more advanced detectors like YOLO (You Only Look Once) could further enhance the system's detection capabilities. YOLO's speed and efficiency in detecting objects might provide a substantial boost to the overall performance of the MOT system.

In summary, while the system has shown commendable results, these proposed enhancements could potentially elevate its performance to match current state-of-the-art trackers.

[4] https://github.com/timmeinhardt/py-motmetrics

References

[1] Yifu Zhang et al. "Fairmot: On the fairness of detection and re-identification in multiple object tracking". In: *International Journal of Computer Vision* 129 (2021), pp. 3069–3087.

[2] Mohamed Elhoseny. "Multi-object detection and tracking (MODT) machine learning model for real-time video surveillance systems". In: *Circuits, Systems, and Signal Processing* 39 (2020), pp. 611–630.

[3] Diego M Jiménez-Bravo et al. "Multi-object tracking in traffic environments: A systematic literature review". In: *Neurocomputing* 494 (2022), pp. 43–55.

[4] Ricardo Pereira et al. "Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics". In: *Applied Sciences* 12 (Jan. 2022), p. 1319. DOI: 10.3390/app12031319.

[5] Xiaolong Wang, Allan Jabri, and Alexei A Efros. "Learning correspondence from the cycle-consistency of time". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2566–2576.

[6] Patrick Dendorfer et al. "Motchallenge: A benchmark for single-camera multiple target tracking". In: *International Journal of Computer Vision* 129 (2021), pp. 845–881.

[7] Guillem Brasó and Laura Leal-Taixé. "Learning a neural solver for multiple object tracking". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 6247–6257.

[8] Anton Milan et al. "MOT16: A benchmark for multi-object tracking". In: *arXiv preprint arXiv:1603.00831* (2016).

[9] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[10] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[11] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. "Learning by tracking: Siamese CNN for robust target association". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2016, pp. 33–40.

[12] Alexandre Alahi et al. "Social LSTM: Human Trajectory Prediction in Crowded Spaces". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 961–971. DOI: 10.1109/CVPR.2016.110.

[13] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. "Tracking without bells and whistles". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 941–951.

[14] Qiankun Liu et al. "GSM: Graph Similarity Model for Multi-Object Tracking." In: *IJCAI*. 2020, pp. 530–536.

[15] Yu Xiang, Alexandre Alahi, and Silvio Savarese. "Learning to Track: Online Multi-Object Tracking by Decision Making". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[16] Siyuan Chen and Chenhui Shao. "Efficient Online Tracking-by-Detection With Kalman Filter". In: *IEEE Access* 9 (2021), pp. 147570–147578. DOI: 10.1109/ACCESS.2021.3124705.

[17] Li Zhang, Yuan Li, and Ramakant Nevatia. "Global data association for multi-object tracking using network flows". In: *2008 IEEE conference on computer vision and pattern recognition*. IEEE. 2008, pp. 1–8.

[18] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[19] Tim Meinhardt et al. "Trackformer: Multi-object tracking with transformers". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 8844–8854.