EasyChair Preprint

№ 10278

Multi-Target Tracking with GPU-Accelerated Data Association Engine

Samiran Kawtikwar and Rakesh Nagi

May 26, 2023

Multi-Target Tracking with GPU-Accelerated Data Association Engine

Samiran Kawtikwar, Rakesh Nagi
Department of Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
{samiran2, nagi}@illinois.edu

***Abstract*—Multi-Target Tracking (MTT) is a challenging problem in the field of data association and sensor data fusion. Many solutions to MTT assume a Markovian nature to the motion of the target to solve the problem and avoid the potential computational complexity. Recently, we have shown that considering a sequence of three time steps and their resulting triplet costs in data association provides a superior solution that better incorporates the kinematic behavior of maneuvering targets. Nevertheless, the triplet costs pose significant computational overhead and scaling challenges. In this paper, we present significant computational advances in a triplet cost-based data association engine for MTT using Graphics Processing Units (GPUs). We achieve this by improving the computational performance of the dual ascent algorithm for dense Multi-Dimensional Assignment Problem (MAP), presented in *Vadrevu and Nagi, 2022*. Our contributions include: (1) A very fast GPU-accelerated Linear Assignment Problem (LAP) solver that solves an array of tiled LAPs without synchronizing with the CPU, (2) Reduction in computational overheads of triplet costs by using gating and compressed matrix representations, and (3) Computational performance studies that demonstrate the effectiveness of our computational enhancements. Our resulting solution is 5.8 times faster than the current solution without compromising the accuracy.**

***Index Terms*—Multi-dimensional Assignment, Data Association, Multi-Target Tracking, Mixed-Integer Linear Programming, Hungarian Algorithm, GPU acceleration, CUDA.**

I. Introduction

The problem of multi-target tracking has been extensively researched using various approaches such as mathematical programming models, Joint Probability Data Association (JPDA), and Multi-Hypothesis Tracking (MHT). In this paper, we employ the Multi-dimensional Assignment Problem (MAP) formulation [1] as a suitable approach for solving the MTT problem. The MAP problem typically consists of a T-dimensional graph with N nodes in each dimension, which is a generalization of the Linear Assignment Problem (LAP) to a *T-partite* graph instead of a *bipartite* graph.

To formulate MTT as a MAP problem we design the following problem setting. The sensors observe targets over a time period divided into multiple time frames, each containing target measurements such as position and velocity. Each target is represented by a node, and each dimension in the graph represents a time frame. The edges in the graph connecting the nodes across time frames represent the cost of matching a node in one time frame to another node in the subsequent time frame. The MAP is then solved to obtain optimal or near-optimal assignments for each node across all T dimensions, with the assignment of each node to another node in adjacent time frames representing the track of the corresponding target.

The optimization formulation presented in this paper allows one to consider the triplet costs resulting from sensor observations in three adjacent time frames between the targets. As discussed in [2], [3], and [4], accounting for the triplet costs has shown better results in terms of tracking accuracy (ITCP and MMEP) and robustness. This is indeed expected as higher-order variables are able to capture more insights of the target trajectory, which is induced into the problem through the objective function coefficients. [5] developed a scaleable dual ascent algorithm that uses GPUs to obtain near-optimal solutions to the MAP. The GPU-accelerated solution presented in [5] to solve MAP has significant room for computational improvement since it solves a general instance of MAP and does not consider optimizations specific to MTT. Additionally, the solution heavily relies on a GPU-accelerated LAP solver developed by [6] which has many CPU-GPU synchronization overheads.

In this paper, we provide a complete and high-performance solution for the MTT problem. Our contributions include the following: (1) A multi-threaded cost generator that simulates the motion of targets and generates accurate costs based on spline interpolation-based techniques with filtering. (2) A very fast GPU-accelerated LAP solver that is free of CPU-GPU synchronization and performs $\sim 500\times$ faster than [6]. (3) A compressed representation of the filtered triplet costs and modification of the dual ascent algorithm to operate on the compressed costs to reduce the computation time by $\sim 5.86\times$ with minimal impact on memory footprint and accuracy.

The rest of the paper is organized as follows: Section II describes some of the previous relevant work in the MTT domain and sets notation. Section III concretely defines the MAP formulation and explains the dual ascent algorithm. Section IV describes our solutions in detail with pseudocodes and parallelization strategies. Section V illustrates the individual performance improvement of each solution and the overall performance improvement. Section VI gives a summary of our contributions, improvements, and scope for future work.

II. Notation and Literature Review

For setting up the MTT problem, we consider a 2-dimensional Region of Interest (ROI), there are N identical targets that randomly move in the region with (approximately) constant velocity. The sensor(s) capture these targets at T distinct times with a fixed frequency, these captures are referred to as *frames*. The captured data gives the coordinates (x, y) of the objects. For the sake of simplicity, we initially assume that the sensor(s) has excellent resolution and there are no missed detections or false alarms. We will see later that our techniques can be adapted to incorporate these realistic issues (similar to [4]). For brevity, we define a set of indices from $\{1, \ldots, L\}$ as $[L]$ where $L \in \mathcal{Z}^+$.

Multiple techniques have been explored to study Multiple Target Tracking (MTT). Some of the significant works in this area include the development of combinatorial models for MTT by Morefield and Mahajan [7], and the use of Joint Probabilistic Data Association Model (JPDA) to solve Multi-Hypothesis Tracking (MHT) by Bar-Shalom and Fortmann [8]. MTT has also been formulated as MAP by [9] using a sliding window model. Additionally, several genetic algorithms have been proposed to solve MTT by [10], [11], [12], and [13]. Since MTT is *fitted* into the MAP framework, the definition of edge costs is always subject to the problem conditions. [14] compared the performance effects of multiple cost definitions based on spline interpolation techniques and it was shown that clamped spline interpolation best captures the motion of maneuverable targets. MAP formulations have been solved for *decomposible costs* by [15] using a Multi Dimensional Assignment with Decomposible Cost (MDADC) formulation. [4] compared the MDADC formulation with a triplet cost-based formulation and showed that the solutions generated with triplet costs are robust to missed and false detections.

MAP is NP-Hard even with triplet costs, exact algorithms become computationally expensive beyond a problem size due to combinatorial explosion. MAP has many polynomial approximation algorithms within a factor 2 of the optimal solution. [16] developed a primal-dual approximation algorithm with theoretical guarantees and developed a solution for multi-target tracking for application in videos. In this paper, we significantly improve the computational performance of the dual ascent algorithm for MAP presented in [5] by developing some general and special optimizations for the MTT problem.

III. Formulation and Solution Technique

We use the formulation established in [17] for the MTT problem. The MTT problem is modeled as a MAP on a T-partite graph where T is the number of frames. Each partition of the graph has N nodes that represent the location of N targets detected by sensor(s). The variables associated with the problem formulation match all the nodes in timeframe t with a node id i. All matchings over T time frames are then interpreted as a track of target i. As explained earlier the objective function of this formulation considers both pairwise and triplet costs. We explain the estimation of these costs in Section II Fig. 1 illustrates a T-dimensional assignment

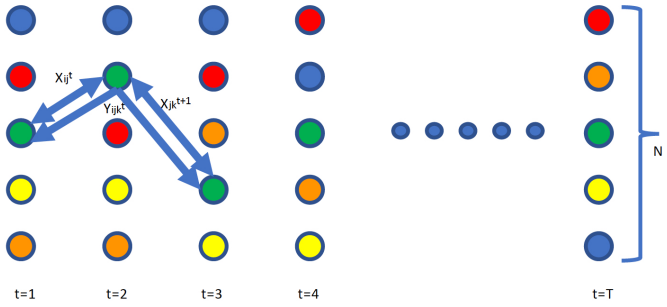

Fig. 1. Multi-Dimensional Assignment Problem Representation

problem. The formulation assumes that there are no missed detections and false alarms, this assumption can be easily relaxed by adding dummy nodes with infinite costs, see [17] for details. The objective of the MAP formulation with pairwise and triplet costs is defined as:

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{T-1} C_{ij}^{p} x_{ij}^{p} + \sum_{i=1}^{N}\sum_{j=}^{N}\sum_{k=1}^{N}\sum_{p=1}^{T-2} D_{ijk}^{p} x_{ij}^{p} x_{jk}^{p+1} \quad (1)$$

Here the binary variable x_{ij}^{p} indicates the matching of node i in time frame p with node j in time frame $p+1$. The formulation has simple assignment constraints for one-to-one matching. This formulation in above form is highly intractable due to its nonlinear nature. Since x_{ij}^{p} is binary, their product can be replaced with a lifted variable $y_{ijk}^{p} = x_{ij}^{p} \times x_{jk}^{p+1}$ with additional linear constraints to relate x and y variables. These additional constraints are then relaxed using Lagrange multipliers to obtain a linear formulation with LAP constraints. We do not go into the details of this relaxation. Interested readers can refer to [17] for a detailed derivation. The relaxed formulation with Lagrange multipliers $(\hat{v}_{jk}^{p})$ is as follows:

LRMAP:

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{T-1} C_{ij}^{p} x_{ij}^{p} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{p=1}^{T-2} (D_{ijk}^{p} - \hat{v}_{jk}^{p}) y_{ijk}^{p} \quad (2)$$

$$\text{s.t.} \sum_{i=1}^{N} x_{ij}^{p} = 1 \quad \forall j \in [N],\ p \in [T-1]; \quad (3)$$

$$\sum_{j=1}^{N} x_{ij}^{p} = 1 \quad \forall i \in [N],\ p \in [T-1]; \quad (4)$$

$$\sum_{k=1}^{N} y_{ijk}^{p} = x_{ij}^{p} \quad \forall i, j \in [N],\ p \in [T-2]; \quad (5)$$

$$\sum_{k=1}^{N} y_{kij}^{T-2} = x_{ij}^{T-1} \quad \forall i, j \in [N]; \quad (6)$$

$$x_{ij}^{p} \in \{0, 1\} \quad \forall i, j \in [N]\ p \in [T-1]; \quad (7)$$

$$y_{ijk}^{p} \geq 0 \quad \forall i, j, k \in [N]\ p \in [T-2]. \quad (8)$$

The dual formulation of the Linear Programming (LP) relaxation of the LRMAP has an interesting structure, we refer to it as DLRMAP. Let $\alpha, \beta, \gamma, \delta$ be the dual variables for constraints (3), (4), (5), and (6) respectively. DLRMAP is given as:

DLRMAP:

$$\max \sum_{j=1}^{N}\sum_{p=1}^{T-1} \alpha_{jp} + \sum_{i=1}^{N}\sum_{p=1}^{T-1} \beta_{ip} \tag{9}$$

$$\text{s.t. } \alpha_{jp} + \beta_{ip} - \gamma_{ij}^{p} \leq C_{ij}^{p}, \quad \forall i,j \in [N]\ p \in [T-2]; \tag{10}$$

$$\alpha_{j(T-1)} + \beta_{i(T-1)} - \delta_{ij} \leq C_{ij}^{T-1}, \forall i,j \in [N]; \tag{11}$$

$$\gamma_{ij}^{p} \leq D_{ijk}^{p} - \hat{v}_{jk}^{p}, \quad \forall\ i,j,k \in [N],\ p \in [T-3]; \tag{12}$$

$$\gamma_{ij}^{(T-2)} + \delta_{jk} \leq D_{ijk}^{T-2} - \hat{v}_{jk}^{T-2}, \quad \forall i,j,k \in [N]; \tag{13}$$

$$\alpha_{jp}, \beta_{ip}, \quad \forall i,j,k \in [N],\ p \in [T-1]; \tag{14}$$

$$\gamma_{ij}^{p} \in \mathbb{R}, \quad \forall i,j \in [N],\ p \in [T-2]. \tag{15}$$

A key observation in the dual formulation is its decomposability into two subproblems, these are referred as X and Y subproblems. From (10), for $\sum_j \sum_p \alpha_{jp} + \sum_i \sum_p \beta_{ip}$ to be maximized, γ_{ij}^{p} has to be large. However by (12), the γ_{ij}^{p} variable has to be constrained according to the following Y subproblem:

$$\Delta_{ij}^{p}(\hat{v}) = \max \left\{ \gamma_{ij}^{p} \middle| \gamma_{ij}^{p} \leq \hat{D}_{ijk}^{p}, \forall\ i,j,k,p \right\},$$

where $\hat{D}_{ijk}^{p} = D_{ijk}^{p} - \hat{v}_{jk}^{p}$.

A. Dual Ascent Algorithm

Based on the structure of X and Y subproblems, a dual ascent algorithm has been developed in [5] and a GPU-accelerated solution was presented. The authors also show that the developed solution is scaleable to multiple CPU nodes with near-linear scaling behavior. As explained in Section I, we significantly improve the performance of this solution. To explain the dual ascent algorithm, we represent the multiplier reduced costs as $\hat{C}_{ij}^{p} := C_{ij}^{p} - \gamma_{ij}^{p}$, and $\hat{D}_{ijk}^{p} = D_{ijk}^{p} - \hat{v}_{jk}^{p}$. Algorithm 1 presents the dual ascent algorithm. The compute upper bound function in Step 5c. of Algorithm 1 can be performed by constructing a feasible solution from the DLRMAP solution and connecting the pairwise assignments of adjacent time steps (x) through the T dimensions. The y variables are simply computed from the x values as $y_{ijk}^{p} = x_{ij}^{p} \times x_{jk}^{p+1}$. Such a solution will not have any conflicts in the y and the x variables. The objective value (2) is computed using this feasible solution. Based on the upper bound (UB) from the feasible solution and lower bound (LB) from DLRMAP, the optimality gap is calculated as: gap $= \frac{\text{UB}-\text{LB}}{\text{UB}} \times 100$.

The dual ascent algorithm consists of 3 main steps: (1) cost transfer (or distribution), (2) solving Y Linear Semi Assignment Problem (LSAP), and (3) solving X LAP. On analyzing the dual ascent algorithm in detail we observed that the main chunk of time is taken by cost transfer and X-LAP operations. Figure 2 shows their relative contribution.

Algorithm 1: RMAP Dual Ascent (RMAP-DA)

1) Cost Initialization:
   a) Initialize $m \leftarrow 0$, $\mathbf{v}^m \leftarrow \mathbf{0}$, $\bar{\nu}(\text{DLRMAP}) \leftarrow -\infty$, and GAP $\leftarrow \infty$.
   b) Initialize cost matrices C and D.
2) Termination: Stop if $m >$ ITN_LIM or GAP $<$ MIN_GAP or LB improvement over last 5 iterations is $< 0.1\%$
3) Cost Distribution and solving the y subproblem:
   a) Update the dual multipliers $(\hat{v}_{jk}^{p})^m \leftarrow (\hat{v}_{jk}^{p})^{m-1} + \lambda_{ijkp}$
   b) Update $\hat{D}_{ijk}^{p} \leftarrow \hat{D}_{ijk}^{p} - (\hat{v}_{jk}^{p})^m, \forall (i,j,k,p)$
4) Solving the y subproblem:
   a) Solve y subproblem and cost coefficients $\hat{D}$.
   b) Let $\Delta_{ij}^{p}(\mathbf{u}^m) \leftarrow \nu(y \text{ subproblem}(i,j,p)), \forall\ (i,j,p)$.
   c) Update $\hat{C}_{ij}^{p} \leftarrow \hat{C}_{ij}^{p} + \Delta_{ij}^{p}(\mathbf{v}^m),\ \forall\ (i,j,p)$.
5) Solve x-LAPs:
   a) Solve T x LAPs of size $N \times N$ and cost coefficients $\hat{C}$.
   b) Update $\nu(\text{DLRMAP}(\mathbf{v}^m)) \leftarrow \nu(x - \text{LAPs})$.
   c) Compute Upper Bound (UB) and GAP.
6) Update $m \leftarrow m + 1$. Return to Step 2.

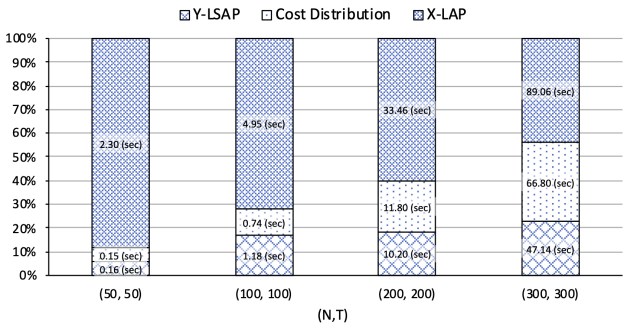

Fig. 2. Relative time for each step in Dual Ascent

IV. Our Solution

In this section, we discuss the details of our high-performance data association engine for the MTT problem. We first discuss the cost generation part and its multi-threaded implementation in detail.

A. Cost Generation with Gating

Scoring is a critical aspect of the solution for models based on mathematical programming. These scores ultimately end up in the objective function and drive the optimal solution. The MAP literature for the MTT problem generally uses Euclidean distances as costs with random noise on an imaginary trajectory. However, the noise is present in the location of the object when it is detected by the sensor. Also, the Euclidean costs represent the shortest distance between the detected positions.

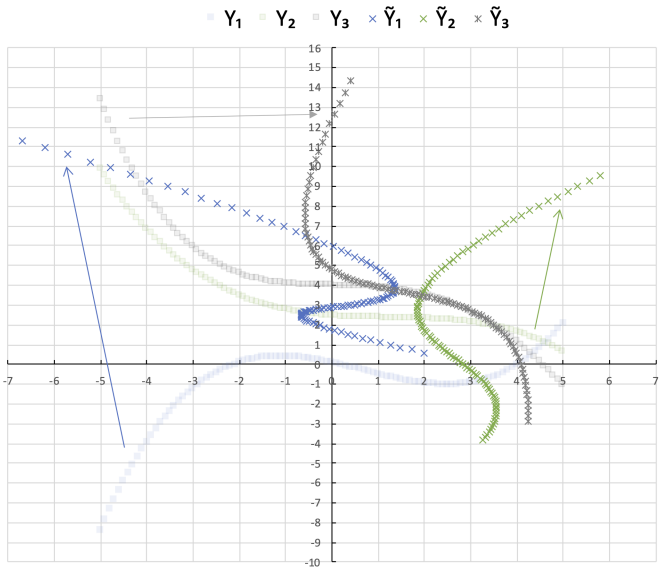

Fig. 3. Generating Particle Trajectories for 3 particles on X-Y grid

In reality, a maneuverable target does not always take a straight path between the targets. Many sensors also detect the heading (velocity direction) of the target. So for a finitely maneuverable object, it is important to consider the path it takes while changing its direction in order to reach the next position. The randomly generated costs based on Euclidean distances fail to benefit from this information. [14] showed that considering clamped spline costs that use the heading data accurately mimic the triplet costs.

Based on these observations, we develop a multi-threaded cost generator for pairwise and triplet costs. We start by generating a polynomial path with randomly generated coefficients. The degree of the polynomial can be specified by the user, we use 3^{rd} and 4^{th}-degree polynomials for our experiments. For a 4^{th} degree polynomial, let the randomly generated coefficients for a specific target be $\{a_0, a_1, a_2, a_3, a_4\}$. Here, a_i represents the coefficient of x^i. To avoid abrupt slopes and turns, we keep reducing the max value of the coefficients (i.e., $E(|a_0|) \geq 4^2E(|a_1|) \geq 4^4E(|a_2|) \geq 4^6E(|a_3|) \geq 4^8E(|a_4|)$) where $E(.)$ is the expectation function. Let the polynomial function generated for target i be $f_i(t)$, where t represents the time. We assume that each target is moving with a constant speed (V_i) which is randomly generated in $U[1, 2]$. Using the speed and timeframes, we find the position and velocity of the targets (y_i and d_{y_i}), using inbuilt tools from the `polynomialsolve` function of *alglib* [18]. This involves finding intersection of a circle with polynomial curve. The method used by *alglib* to find this intersection is out of the scope of this paper. Once y_i and d_{y_i} are generated, we rotate each point along the origin by a randomly generated angle $\theta \in U(-\Pi, \Pi])$ and translate by a randomly generated distance $d \in U(0, 2.5)$. We represent the modified positions and velocities by $\tilde{y_i}$ and $\tilde{d_{y_i}}$. We represent this process in Fig. 3. After generating $\tilde{y}$, we add noise to the points using a Gaussian random generator.

We start scoring once $(\tilde{x}, \tilde{y})$ and $(\tilde{d_x}, \tilde{d_y})$ are generated for each target at every timeframe. Algorithm 2 shows the recipe and the parallelization strategy. Here, the `Spline-Fit` and `Curve-Dist` functions are taken from *alglib* [18]. Gating is introduced in y-costs to reduce their computational footprint. Since Euclidean distance is the shortest distance between two points, the target is unlikely to take a path where the distance is higher than how much it can travel. To avoid aggressive filtering, we relax the constraints by a factor of ϵ. ϵ needs to be tuned to achieve a desired level of filtering, selecting ϵ depends on the quality of data and the noise in the sensor readings. We observe that the dual ascent algorithm is not very sensitive to wrong values of ϵ and always solves to near optimality. Scoring can also be accelerated on GPUs since it is embarrassingly parallel. However, CUDA does not have a well-documented library for interpolation and curve intersection (like *alglib*).

Algorithm 2: Gated Scoring

Input: N ← Num Targets,
T ← Num Frames,
δt ← Time between each frame (1/frequency)
M: Big Number (10^6)
$(x_i, y_i, d_{x_i}, d_{y_i}), V_i \ \forall i \in N$
Output: Pair costs X_{costs}, Triplet costs Y_{costs}
`// Initialize vectors`
$Y_{costs} \leftarrow \phi,\ X_{costs} \leftarrow \phi$;
`// Parallel for each` i
for $i, j \leftarrow$ *1* **to** N **do**
  for $k \leftarrow$ *1* **to** N **do**
    for $t \leftarrow$ *1* **to** $T - 2$ **do**
      if `Eucli-Dist` $(i, j) < V_i \times \delta t \times \epsilon$ *and* `Eucli-Dist` $(j, k) < V_i \times \delta t \times \epsilon$ **then**
        $Y_{costs}\ (i, j, k, t)$ = `Abs-Spline-Dist` (i, j, k);
      else
        $Y_{costs}\ (i, j, k, t) = M +$ `Abs-Spline-Dist` (i, j, k);
  for $t \leftarrow$ *1* **to** $T - 1$ **do**
    $X_{costs}(i, j, t)$=|`Eucli-Dist` (i, j) -$V_i \times \delta t$ |
def `Abs-Spline-Dist` (i, j, k)**:**
  Curve = `Spline-Fit` $(x_i, x_j, x_k, y_i, y_j, y_k, d_{x_i}, d_{x_j}, d_{x_k}, d_{y_i}, d_{y_j}, d_{y_k})$;
  return |`Curve-Dist` (i, j) |-|`Curve-Dist` (j, k) |

B. Block LAP solver for X-LAP

The dual ascent algorithm relies on the LAP structure of the X subproblem. It can be clearly seen from Algorithm 1 that the X subproblem contains T disjoint LAPs. The implementation by [5] uses a tiled LAP solver developed by [14]. Following a detailed computational analysis, we found that this LAP solver has too many CPU-GPU synchronization barriers

which tremendously slows down performance. GPUs offer massive parallelism but their performance can be constrained by the Single Instruction Multi-Data (*SIMD*) architecture. The thread hierarchy of a GPU consists of thread blocks that are synchronizable. Each GPU has several 100s of thread blocks that are resident in a Streaming Multiprocessor (SM) and each thread block has several 1000s of threads. For Ex. the NVIDIA A100 GPU has 108 SMs and each SM can concurrently handle a maximum of 2048 threads. The SMs can be thought of as individual computational hardware in the GPUs, synchronizing within the SM (in particular, the thread block) has low overhead while synchronizing across SMs is latency bound. A usual but inefficient strategy to synchronize across SMs is to terminate the GPU kernel and launch another one for the next operation which causes significant overheads.

To tackle this, we implemented a Block LAP solver. The implementation details of this solver are out of the scope of this paper. Interested readers are requested to refer to [19] for a detailed explanation of a similar approach. The important observation here is that the Block LAP solver has no synchronization overheads with the CPU and is well-balanced in terms of task distribution per thread. As the name suggests, the Block Hungarian solver uses one thread block to solve an LAP. With this structure, the threads in a single thread block concurrently perform similar operations which is suitable to the *SIMD* architecture. Block Hungarian also performs many efficient operations to improve the computational performance. We gain tremendous performance improvement by replacing the tiled LAP solver with block Hungarian.

C. Compressed Cost Distribution

It is clear from Fig. 2 that the second most time-consuming operation of the dual ascent algorithm is cost distribution. This operation is embarrassingly parallelizable, but the computational complexity $O(N^3T)$ of this problem makes it expensive. Algorithm 3 gives the implementation of the naive function. We can see that there are $O(N^2T)$ threads launched. For a problem with 100 targets and 100 time frames, this function would need around $1M$ threads, with each thread performing 100 operations. Following a careful inspection of the dual ascent algorithm, we can see that the Lagrange multipliers are upper bounded by X_{costs}. The X_{costs} are only updated by δ (see step (4b) in Algorithm 1) which is $min_k\ y_{(i,j,k,p)}$. Since gating ensures that each particle will have some finite triplet cost, the minimum is bounded by $max\ Y_{costs} : Y_{costs}! = M$. Hence, the Lagrange multipliers are significantly small compared to M. Hence, we can totally avoid the cost distribution operation for $Y_{costs} : Y_{costs} \geq M$.

In this passage, we discuss the techniques used to avoid the cost transfer operation for $Y_{costs} \geq M$. We first use a compressed format for storing the Y_{costs} array, this format is similar to the Compressed Sparse Coordinate (*CSRCOO*) format which is widely used to represent sparse matrices. Fig. 4 shows the representation. We use this representation and store all the index values (i, j, k, t). On experimenting with the dataset, we observed that gating reduces almost 80% of

Algorithm 3: Naive Cost Distribution

Input: N ← Num Targets,
T ← Num Frames,
Y_{costs}, V ← Lagrange Multipliers
// Launch $N^2 \times T$ threads
t =threadIdx.y
j =threadIdx.x%N
k =threadIdx.x/N
for i *in* N **do**
  $Y_{costs}\ (i,j,k,t) = Y_{costs}\ (i,j,k,t)$ - $V\ (j,k,p)$;

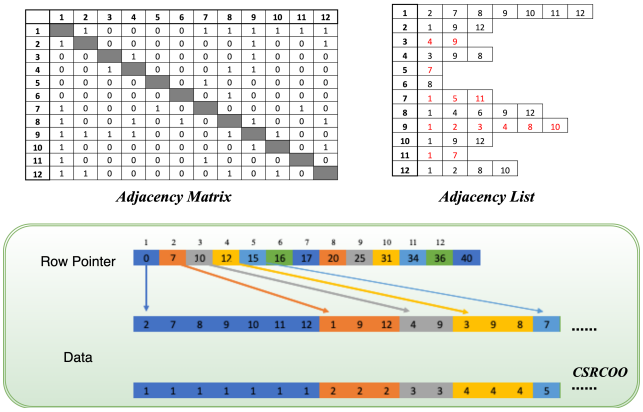

Fig. 4. Compressed Sparse Coordinate representation

the values. We store this compressed dataset along with the original Y_{costs}. Note that this compression needs to be done only once while generating the data. Post compression we can use an algorithm similar to Algorithm 3 but only launch threads for the entries with $Y_{costs} \leq M$. Since the threads residing in the same block are SIMD, we further improve this approach by launching N^2 thread blocks each with 32 threads. Each thread block picks a specific chunk of entries from Y_{costs} and processes them using the compressed index array. Algorithm 4 illustrates the recipe. Note, this algorithm performs around 80% fewer computations and is more suited to the SIMD architecture.

Algorithm 4: Compressed Cost Distribution

Input: N ← Num Targets,
T ← Num Frames,
Y_{costs}, V ← Lagrange Multipliers,
$RowPointer$ ← From CSRCOO.
// Launch $N * T$ thread blocks (32 threads each)
k =blockIdx.x % N
t =blockIdx.x / N
for $p =$ threadIdx.x; $p < |RowPointer\ (k,t)|$; $p+ = 32$ **do**
  (i, j) = $RowPointer\ (k,t)[p]$;
  $Y_{costs}\ (i,j,k,t) = Y_{costs}\ (i,j,k,t)$ - $V\ (j,k,t)$;

V. Computational Experiments

We perform computational experiments to evaluate the performance of our solutions. We also perform incremental analysis to provide the isolated performance impact of each solution. The baseline and our solutions are compared on the same computational platform. These are the main results in this section:

1) The impact of gating on MMEP and ITCP scores
2) MMEP scores with Euclidean versus Spline costs
3) Time speedups with block LAP
4) Time speedups with compressed cost distribution

A. Experimental Setup and Problem Data

This section's findings were generated using a high-performance computing cluster named "delta", which is equipped with a dual-socket 64-core AMD EPYC 7763 ("Milan") CPU and NVIDIA A100 40GB HBM2 GPU. The cluster has 256 GB of DDR4 memory, and the program was compiled with NVCC (CUDA 11.7, driver version 515.65) SM version 80, and GCC 11.2 with the O3 flag. To ensure an equitable evaluation, all baseline measurements were also taken on the same hardware platform using comparable compilation flags.

We conducted experiments on problem sizes where $5 \leq N \leq 300$ and $5 \leq T \leq 300$. We handle the pairwise and triplet costs with double precision (64 bits). For small problems where $N, T \leq 50$, we generated five different cost instances and reported the average performance to prevent bias. However, for large problems ($N, T \geq 50$), only one instance was generated.

B. Evaluation Criteria

To evaluate the algorithm's performance on the simulated data from our cost generator, we computed the Incorrect Trajectory Count Percentage (ITCP) and Mismatch Error Percentage (MMEP) scores [20]. These scores measure the error between the original trajectories and the algorithm's solutions. We defined an incorrect trajectory as one with at least one incorrect assignment. The ITCP score computes the percentage of incorrect trajectories out of the total number of trajectories. The MMEP score computes the percentage of total mismatch errors (MME) in all trajectories over all the assignments. Since the total number of assignments in MTT $N \times (T-1)$, we can compute the MMEP score as MMEP $= \frac{\sum \text{MME}}{N \times (T-1)} \times 100$.

Both the ITCP and MMEP measures are essential in evaluating the quality of a solution. While MMEP is more focused on the assignment level, ITCP provides an overall sense of the quality of trajectories. A solution with a high MMEP but a low ITCP score can still be a good solution. For example, if two trajectories are close together, leading to multiple mismatched assignments, the MMEP score would be high, but the ITCP score would be low as only two trajectories are incorrect overall. Conversely, if all trajectories are close together, at least one error is reported in each trajectory and not many errors overall, the ITCP score would be high and the MMEP score would be low.

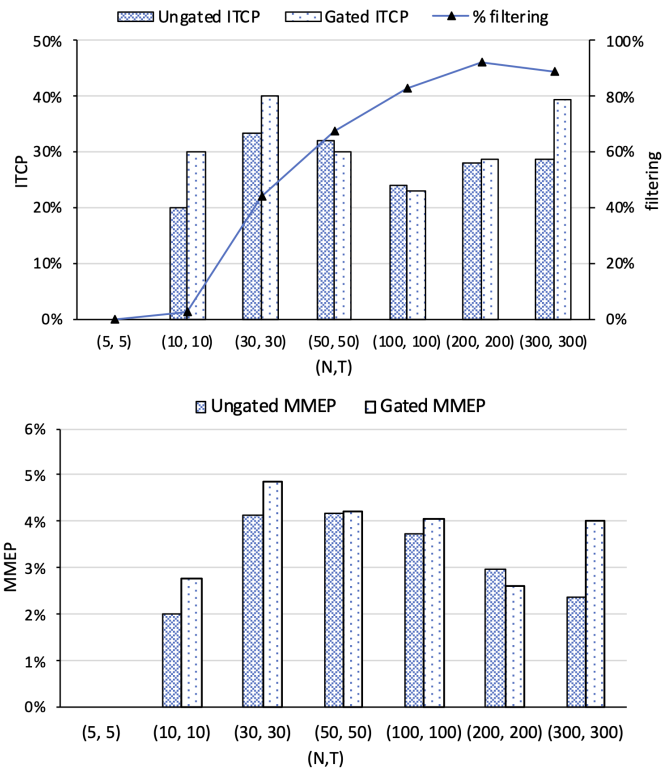

Fig. 5. Effect of Gating on MMAP and ITCP scores

We use time speedups to report the performance of our solution. Speedup is defined as the ratio of the runtime of our solution to the baseline when using same computational environment. Following the norms in the HPC literature, we use geometric mean of the speedups to report aggregate performance benefit.

C. Results

The first part of our solution included filtering the triplet costs based on gating and constant velocity assumption. The filtering, if done too aggressively may result in a poorer quality of solution. For these experiments, we simulated filtered and unfiltered costs. For filtering with gating, we considered the relaxation factor $\epsilon = 5$. To remove other variations, we run both problems for 100 Dual Ascent iterations. Fig. 5 shows the impact of gating. The MMEP score usually increases after gating, the effect is smaller for large problems. This figure establishes that gating has minimal impact on the quality of the solution.

The second part of our results shows the significant improvement in LAP solution times. The plots had to be drawn on a log scale since the speedups were so high. We only show results for large problems as smaller problems take insignificant time for both cases. We achieve a maximum time speedup of $1001\times$ and a geometric mean speedup of $708.84\times$ across problem sizes ranging from $[50, 50]$ to $[300, 300]$.

The third part of our results shows a significant improvement in the cost transfer part of the dual ascent algorithm. This

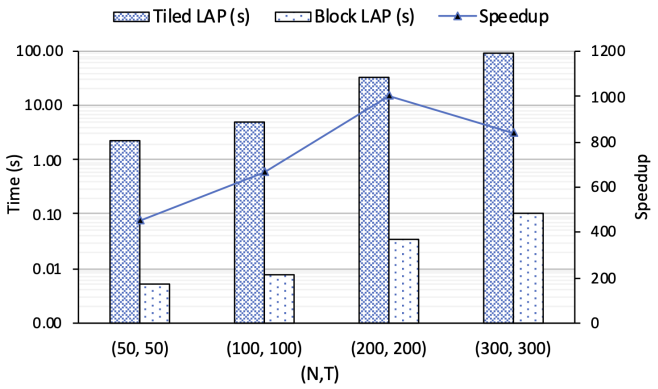

Fig. 6. Speedups with block LAP (GeoMean: $708\times$)

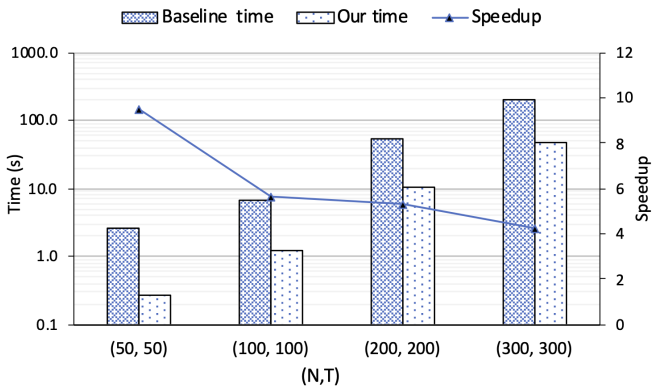

Fig. 8. Overall time speedups (GeoMean: $5.86\times$)

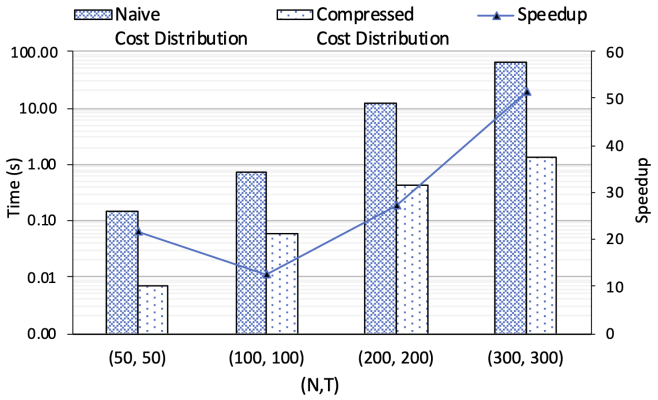

Fig. 7. Speedups with compressed cost distribution (GeoMean: $24.7\times$)

plot is also drawn on a log scale. Since the solution employed reduces the computational complexity of cost transfer, we see increasing speedup with increasing problem sizes. The overall geometric mean speedup from this solution is $24.72\times$ across problem sizes ranging from $[50, 50]$ to $[300, 300]$.

Finaly, we present the overall performance improvements in the dual ascent algorithm for the same number of iterations. We establish that the quality of the solutions is not deteriorated by these optimizations.

VI. Conclusions and Further Work

In this paper, we carefully analyzed the GPU-accelerated dual ascent algorithm for the multi-dimensional assignment problem and significantly improved its performance. We used insights from the MTT problem structure, GPU parallelization literature, and compressed representations to develop a high-performing solution. We achieved a geometric mean speedup of $5.89\times$ on the overall runtime of the GPU-accelerated dual ascent algorithm. We performed a thorough computational analysis of the GPU-accelerated dual ascent algorithm and found that almost 85% of the run time is consumed by solving the X-LAPs and cost distribution. For reducing the X-LAP solution times, we developed a fast block LAP implementation that does not require CPU-GPU synchronizations. This block LAP solver gives a geometric mean speedup of $708\times$ over [6]. For reducing the computational complexity of the cost distribution function, we used gating and filtered almost 85% of triplet costs (Y_{costs}). We used MMEP and ITCP evaluation metrics to establish that the filtering does not significantly deteriorate the solution quality. Post filtering, we used a *CSRCOO* data structure from the sparse matrix literature to efficiently implement the cost distribution function. This resulted in a geometric mean speedup of $24.7\times$. Overall these two contributions reduced the total time by a factor of 5.8.

The current performance is bounded by the Amdahl's law [21]. Since we only improved the performance of functions that take 85% of total time, the overall performance improvement would be bounded by $\frac{100}{15+\delta}$. Here, δ is the overall speedup in functions that take 85% of the time. Hence, further work should be focused towards improving the performance of the remaining tasks (which took 15% of total time). i.e., solve Y-LSAP and dual update, see Fig. 2. The memory footprint of the Y_{costs} is still very high at $O(N^3T)$, so it should be avoided to fully store the matrix. Since [5] showed that the dual ascent algorithm for MAP is easily scaleable with good characteristics, we can easily increase the size of this problem and achieve even more speedups.

Finally, all the pairwise and triplet scores considered in this paper are deterministic. Some mainstream tracking algorithms like [22] and [23] use probabilistic methods to find expected matching scores. It would be interesting to fuse these scoring techniques into our framework and examine the quality of the resulting solution.

Acknowledgments

This research used the Delta advanced computing and data resource, which is supported by the National Science Foundation (award OCI 2005572) and the State of Illinois. Delta is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. We also acknowledge NVIDIA for an equipment gift under the Applied Research Accelerator Program. The authors would like to thank the anonymous reviewers for their helpful reviews and comments.

REFERENCES

[1] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems: Revised Reprint*. Society for Industrial and Applied Mathematics, Jan. 2012. [Online]. Available: http://epubs.siam.org/doi/book/10.1137/1.9781611972238

[2] R. T. Collins, "Multitarget data association with higher-order motion models," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 1744–1751.

[3] A. A. Butt and R. T. Collins, "Multiple target tracking using frame triplets," in *Computer Vision – ACCV 2012*, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 163–176.

[4] O. Reynen, S. Vadrevu, R. Nagi, and K. LeGrand, "Large-scale multi-dimensional assignment: Problem formulations and GPU accelerated solutions," in *Proceedings of the 22nd International Conference on Information Fusion*, July 2019, pp. 1–8.

[5] S. Vadrevu and R. Nagi, "A GPU Accelerated Dual-Ascent algorithm for the Multidimensional Assignment Problem in a multitarget tracking application," *IEEE Transactions on Automation Science and Engineering*, pp. 1–15, 2022.

[6] K. Date and R. Nagi, "GPU-accelerated hungarian algorithms for the linear assignment problem," *Parallel Computing*, vol. 57, pp. 52–72, 2016.

[7] C. Morefield, "Application of 0-1 integer programming to multitarget tracking problems," *IEEE Transactions on Automatic Control*, vol. 22, no. 3, pp. 302–312, 1977.

[8] Y. Bar-Shalom and X.-R. Li, *Multitarget-multisensor tracking: Principles and techniques*. YBs London, UK:, 1995, vol. 19.

[9] T. Kirubarajan, H. Wang, Y. Bar-Shalom, and K. R. Pattipati, "Efficient multisensor fusion using multidimensional data association," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 37, no. 2, pp. 386–400, April 2001.

[10] Zne-Jung Lee, Shun-Feng Su, and Chou-Yuan Lee, "Efficiently solving general weapon-target assignment problem by genetic algorithms with greedy eugenics," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 1, pp. 113–121, Feb 2003.

[11] F.-X. Liu and Q.-H. Xing, "Optimized target assign method based on mixed genetic algorithms," vol. 22, 07 2002.

[12] W. Wang, S.-C. Cheng, and Y.-Z. Zhang, "Research on approach for a type of weapon target assignment problem solving by genetic algorithm," vol. 30, pp. 1708–1711, 09 2008.

[13] J.-X. Yu, S.-H. Wang, and W.-X. Cheng, "Target allocation decision making based on improved genetic algorithms with local search," vol. 30, pp. 1114–1117+1162, 06 2008.

[14] K. Date and R. Nagi, "Tracking multiple maneuvering targets using integer programming and spline interpolation," in *Proceedings of the 21st International Conference on Information Fusion*, 9 2018, pp. 1293–1300.

[15] S. Natu, K. Date, and R. Nagi, "GPU-accelerated lagrangian heuristic for multidimensional assignment problems with decomposable costs," *Parallel Computing*, vol. submitted, 2019.

[16] S. S. Srinivasan, A. Anand, and A. Divakaran, "Approximate multidimensional assignment for multi-object tracking," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6767–6776.

[17] S. Vadrevu and R. Nagi, "A dual-ascent algorithm for the multi-dimensional assignment problem: Application to multi-target tracking," in *Proceedings of the IEEE 23rd International Conference on Information Fusion (FUSION)*, 2020, pp. 1–8.

[18] T. A. Project, "ALGLIB - a numerical analysis and data processing library," 2023. [Online]. Available: http://www.alglib.net/

[19] P. A. Lopes, S. S. Yadav, A. Ilic, and S. K. Patra, "Fast block distributed CUDA implementation of the Hungarian algorithm," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 50–62, Aug. 2019, block Hungarian Paper.

[20] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the CLEAR MOT metrics," *Journal on Image and Video Processing*, vol. 2008, p. 1, 2008.

[21] D. P. Rodgers, "Improvements in multiprocessor system design," in *Proceedings of the 12th Annual International Symposium on Computer Architecture*, ser. ISCA '85. Washington, DC, USA: IEEE Computer Society Press, 1985, p. 225–231.

[22] A. Logothetis, V. Krishnamurthy, and J. Holst, "On maneuvering target tracking via the PMHT," in *Proceedings of the 36th IEEE Conference on Decision and Control*, vol. 5. San Diego, CA, USA: IEEE, 1997, pp. 5024–5029. [Online]. Available: http://ieeexplore.ieee.org/document/649857/

[23] C. Rago, P. Willett, and R. Streit, "A comparison of the JPDAF and PMHT tracking algorithms," in *Proceedings of the 1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 5. Detroit, MI, USA: IEEE, 1995, pp. 3571–3574. [Online]. Available: http://ieeexplore.ieee.org/document/479758/

Samiran Kawtikwar is a Ph.D. student at the University of Illinois at Urbana Champaign. His research interests lie at the intersection of High-Performance computing and Large-Scale optimization. His current research primarily focuses on developing GPU-accelerated solutions for NP-Hard optimization problems.

Rakesh Nagi (Senior Member, IEEE) is a Willett Professor with the Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana–Champaign. He has more than 200 journal and conference publications. His major research thrusts are in the area of production systems, applied/military operations research, and data fusion using graph theoretic models.