# MODELING MULTIPLE-OBJECT TRACKING AS CONSTRAINED FLOW OPTIMIZATION PROBLEM

## 1 Introduction

Solving optimization problems through greedy algorithms may cause the system to miss global optimum solution. On the other hand, global optimum solution may require NP-hard algorithm. Object tracking is an interesting problem that can be solved by independent detection of objects in individual frames of a video sequence and then linking these detections into possible trajectories. This approach seems very robust and can easily deal with spurious or missed detections by incorporating information from neighboring video frames. For visually similar multiple objects, linking detections into possible trajectories is an optimization problem of high complexity. Greedy dynamic approach for this task ensures a speedy solution but does not guarantee a global optimal solution. This task can be formulated as an instance of integer linear program (ILP) with a few a-priori restrictions such as fixed number of objects throughout the video [1]. ILP formulation ensures global optimal solution but its solution is NP-Complete problem. The main focus of this report is a research paper which presents a way to relax a-priori conditions while formulating the problem as an instance of ILP [2]. This work also describes how the problem formulation can be relaxed to continuous linear program (LP) whose solvers have Polynomial time average complexity. Still the problem is too complex even for a reasonable number of objects being tracking in a relatively short video sequence. Therefore, the problem is further modified as an instance of k-path optimization problem which makes the solution scale well to practical applications.

## 2 Multiple Object Tracking as Integer Linear Program

This part studies the formalization of multi-target tracking problem in terms of an integer programing. To begin with, the physical area is divided into $k$ discrete locations (cell) where for each cell $N\ (k)$ shows the indices of the neighborhoods of $K$. Particularly, if an object was in location $K$ in time $t$, then $N(k)$ shows the possible location can be reached by the object in time $t + 1$. By this definition, the

occupancy over the time can be modeled using a directed graph where $K_t$ (location $K$ in time frame $T$) is a vertex and $e_{i,j}^t$ is an edge defined form node $(i,t)$ to node $(j,t+1)$ if and only of $j \in N(i)$. Furthermore, there is an edge $e_{i,i}^t$ in the case an object stays static between time frame $t$ and $t+1$. There are also labels $m_j^t$ and $f_{i,j}^t$ assigned to each vertex and edge respectively where $m_j^t$ shows the number of objects located in node $i$ in time $t$ and $f_{i,j}^t$ indicates the number of objects moving from location $i$ to $j$ between time frames $t$ and $t+1$. This concept is illustrated in figure 1.

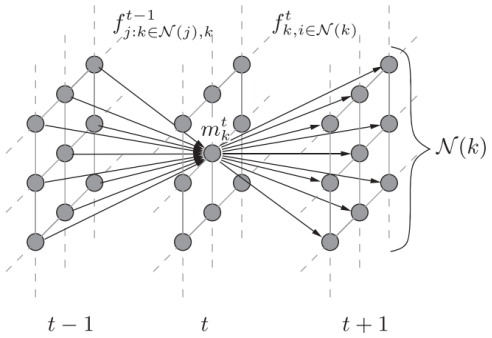


*Figure 1. A flow model used for tracking objects moving on a 2D grid, such as in pedestrian tracking.*

By this definition, the sum of flows entering each vertex $j$ from $t-1$ in equal to vertex label $m_j^t$ and equal to sum of flows leaving the node $j$ in time $t$. This is stated in the following equation.

$$\forall j,t \sum_{i:j\in N(i)} f_{i,j}^{t-1} = m_j^t = \sum_{k\in N(j)} f_{j,k}^t \qquad (1)$$

However, the constraint applied to this approach is that each location (node) can be occupied by only one object at a time. In other words, the upper bound for sum of flows that can leave a location (or arrive to a location) in time $t$ is 1. Moreover, flows are nonnegative. These two constraints are formulated in (2) and (2) as follows:

$$\forall k,t, \quad \sum_{j\in N(k)} f_{k,j}^t \leq 1 \qquad (2)$$
$$\forall k,j,t, \quad f_{k,j}^t \geq 0 \qquad (3)$$

Additionally, the number of tracked objects may change since objects can enter to the tracking area or even leave this area. Thus to model that, it requires to add two virtual nodes $v_{source}$ $and$ $v_{sink}$ (named virtual nodes since they are not real location in the scene) to the graph and connect them to the entrance/exit locations such as borders of the scene and doors. Addition to that, $v_{source}$ should be linked to all nodes in the first frame and nodes in the last frame should goes to $v_{sink}$. This is shown in figure 2.

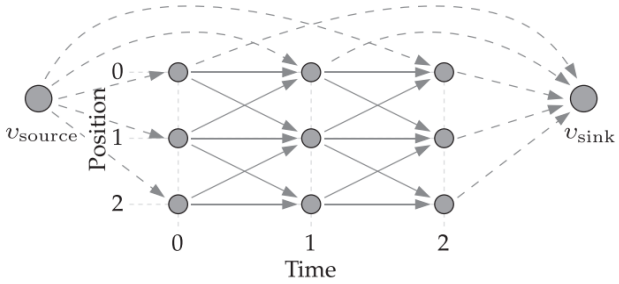


*Figure 2. A complete flow system for a simple graph consisting only of three positions and three time frames. Here, we assume that position 0 is connected to the virtual positions, and therefore is a possible entrance and exit point. Flows to and from the virtual positions are shown as dashed lines.*

This concept states a new constraint where sum of all flows entering the area is equal to sum of those leaving the area.

$$\underbrace{\sum_{j \in \mathcal{N}(v_{\text{source}})} f^t_{v_{\text{source}},j}}_{\text{Leaving } v_{\text{source}}} = \underbrace{\sum_{k:v_{\text{sink}} \in \mathcal{N}(k)} f^t_{k,v_{\text{sink}}}}_{\text{Arriving at } v_{\text{sink}}} . \qquad (4)$$

By using the object detector to process the sequences, the value of each location $i$ in time $t$ can be estimated. Hence, using this estimation and $M_i^t$ as the random variable presenting the true value of $i$ in time $t$, the posterior probability of an object in location $i$ can be calculated as follows:

$$\rho_i^t = \hat{P}\left(M_i^t = 1 \middle| I^t\right) \qquad (5)$$

Where $I^t$is the signal, such as sequence of pictures is taken with different cameras from a same scene, in time $t$. Using the equation 5 and variables $m_j^t$, the occupancy map $m$ (holding variables $m_j^t$) can be constructed where there exists a sets of flows $f^t_{k,j}$ that satisfied constraints in (1) to (4). The set of these feasible maps called $\mathbb{F}$. Therefore, the final goal of this study is solve the following equitation.

$$m^* = arg\,max_{m \in \mathbb{F}} \hat{P}(M = m | I^t) \qquad (6)$$

By assuming $M_i^t$, given $I^t$, as conditional independence variable, the equation can be formulated as it is stated in (7).

$$m^* = arg\,max_{m \in \mathbb{F}} log \prod_{t,i} \hat{P}\left(M_i^t = m \middle| I^t\right) = arg\,max_{m \in \mathbb{F}} \sum_{t,i} log \hat{P}\left(M_i^t = m \middle| I^t\right) \qquad (7)$$

Since value of $m_i^t$ is either zero or one, so:

$$m^* = arg\,max_{m \in \mathbb{F}} \sum_{t,i} (1 - m_i^t) log \hat{P}\left(M_i^t = 0 \middle| I^t\right) + m_i^t log \hat{P}\left(M_i^t = 1 \middle| I^t\right) \qquad (8)$$

By cutting out $log \hat{P}\left(M_i^t = 0 \middle| I^t\right)$ as a term that does not depend on $m$, the equation can be simplified as:

$$m^* = arg\,max_{m \in \mathbb{F}} \sum_{t,i} m_i^t log \frac{\hat{P}\left(M_i^t = 1 | I^t\right)}{\hat{P}\left(M_i^t = 0 | I^t\right)} = arg\,max_{m \in \mathbb{F}} \sum_{t,i} \left( log \frac{\rho_i^t}{1 - \rho_i^t} \right) m_i^t \qquad (9)$$

Finally, the variable $m^*$ is defined as the linear expression of $m_i^t$. Thus, the equation can be translated into the integer programming where:

$$\begin{aligned} \textit{Maximize} \quad & \sum_{t,i} \left( log \frac{\rho_i^t}{1-\rho_i^t} \right) \sum_{j \in N(i)} f_{i,j}^t \qquad (10) \\ \textit{Subject to} \quad & \forall i,j,t \quad f_{i,j}^t \geq 0 \\ & \forall i,t \sum_{j \in N(i)} f_{i,j}^t \leq 1 \\ & \forall i,t \sum_{j \in N(i)} f_{i,j}^t - \sum_{k:i \in N(k)} f_{k,i}^{t-1} \leq 0 \\ & \sum_{k \in N(v_{source})} f_{j,k}^t - \sum_{i:j \in N(i)} f_{i,j}^{t-1} \leq 0 \end{aligned}$$

Using this formula, the stated integer programming can be solved using any generic LP solver. However, because the complexity of LP solver is NP-complete, using these solvers in a large size problems (like the one here) is not practicable. One solution to this issue is to use relation methods and solve a continuous Linear Program instead. In the following section using the linear programming in multiple objects tracking is studied.

# 3 Multiple Objects Tracking as Linear Program

Though the representation of multiple objects tracking problem as an instance of integer linear program (ILP) provides a valid mechanism to obtain global optimum solution, solution of integer linear program is an *NP-complete* problem. Thus a global optimum solution of multiple-object tracking seems to be beyond the limits of practical computational time.

## 3.1 Continuous Linear Program

A continuous linear program solver, on the other hand, has Polynomial complexity on average. It is intuitive to try to relax 'integer' condition of integer linear program representation of multiple object tracking problem and achieve an instance whose solution is possible within in Polynomial time. There are simplex-based algorithms with average case Polynomial complexity to solve an LP.

### 3.1.1 Simplex Based LP Solution

The simplex method was presented by G.B. Dantzig in 1949. The classical reference is his monograph from 1963 [3]. Solution of a system $Cx \leq b; for\ x \geq 0$ lies at the bounrady of *polytope* of solutions of the form $\{x \mid Cx \leq b, x \geq 0\}$. For non-degenerate cases, only one set of vertices represent optimal solution of the system. In degenerate cases, there are multiple solutions. Even in degenerate case, at least two solutions are from the vertices of *polytope*. The simplex algorithm traverses on the *polytope*

underlying the linear program from vertex to vertex along the edges until an optimal vertex is reached [4].

In Dantzig's algorithm, *slack* variables ($w_i \; for \; i = 1,2, \dots n$) are introduced to convert inequalities to equalities. For a system

$$maximize \sum_{j=1}^{n} d_j x_j$$
$$subject\ to \sum_{j=1}^{m} c_{ij} x_j \leq b_i \quad i = 1,2 \dots n$$
$$x_j \geq 0 \quad j = 1,2, \dots . m$$

Equalities are formed as

$$\delta = \sum_{j=1}^{m} d_j x_j$$
$$b_i - \sum_{j=1}^{m} c_{ij} x_j = w_i \quad i = 1,2 \dots n$$

The algorithm is iterative. It starts with one solution of original variables which satisfies the equations and non-negativity condition and then iteratively look for a better solution. Having ties in choices to be made at each iteration and cycling are rare in real world problems. It is generally believed that LP problems in real world are disinclined to be *degenerate* [4].

Klee and Minty created an artificial class of LP problem in 1972 for which simplex algorithm has to pass all $2^n$ possible vertices, thus proving that worst-case performance of simplex algorithm is not Polynomial [4]. But average case complexity is believed to be Polynomial which is an improvement over having to solve an NP-complete problem in the form of ILP.

### 3.1.2 Optimal Solution of LP

The catch is that global optimum solution of the modified continuous linear program is, in general, not the same as global optimum solution of original integer linear program. Thus, there seem to be a trade-off between achieving a global optimum solution and time complexity of the system.

The particular problem that this paper discusses [2], has a specific property that solves the problem of mismatch between global optimum solution of continuous and integer linear program. That specific property is called 'Total Unimodularity' and constraint matrix defined previously while presenting multiple objet tracking as an instance of integer linear program possesses this property.

## 3.2 Total Unimodularity

For a linear program of the form $x \leq b$ ; $for\ x \geq 0$ , a specific property of matrix $C$ , called total-unimodularity, ensures that optimum solution obtained is in the form of integers. Thus optimal solution of continuous linear program and original integer linear program is the same.

### 3.2.1 Totally Unimodular Matrix

"A rectangular matrix has total-unimodularity if all its square sub-matrices have determinants belonging to set $\{-1,0,1\}$."

If the matrix has this property, there are two theorems that hold for the linear program and its constraint matrix.

Theorem 1: A square matrix $C$ of size $m \times n$ is totally unimodular if and only if for every subset of its rows i.e. $R \subseteq \{1,2,3 \dots m\}$ , threre exists a partition $R = R1 \cup R2$ and $R1 \cap R2 = \emptyset$ such that

$$\forall j = 1,2,3 \dots n \sum_{i \epsilon R1} a_{ij} - \sum_{i \in R2} a_{ij} \in \{-1,0,1\}$$

Theorem 2: If square matrix $C$ is totally-unimodular, the vertices of the polytope $\{x \,|Cx \leq b, \geq 0\}$ have integer values for any integral $b$.

The vertices of this polytope represent the optimal solution of the linear program. Thus the linear program has integral optimal solution even without the condition of integral solution being embedded into the program formulation. The authors of the papers have provided a proof for the total unimodularity of the constraint matrix for the linear program representing multiple object tracking problem [2]. This proof clearly implies that even after relaxing the 'integer' condition of the original integer linear program representation of multiple object tracking problem, the optimal solution obtained by LP solvers with Polynomial average complexity is the same as that of original representation. First we will discuss a simple proof of how total unimodularity ensures integral solutions for the linear program.

### 3.2.2 Integral Solution Of Linear Program

Consider a quadratic system $Cx = b$, and $C_j = (C_1, C_2, \dots . C_{j-1},\ b\ ,\ C_{j+1}, \dots\ C_m)$ representing a $C$ matrix with j[th] columns replaced with $b$. The Cramer's rule gives the following solution for the system

$$x_j = \frac{|C_j|}{|C|}\ , where\ |M| denotes\ the\ dterminant\ of\ matrix\ M$$

If we denote the matrix created by deleting i[th] row and j[th] column of $C$ as$C_{ij}$, then we can express determinant of $C_j$ as following

$$|C_j| = \sum_i (-1)^{i+j}\ b_i\ |C_{ij}|$$

We have already established from the definition of total unimodularity that if a matrix $C$ has the property of total unimodularity, then for all of its square sub-matrices $|C_{ij}| \epsilon \{-1,0,1\}$. Thus the solution $x$ will be integral as long as $b$ is an integral vector. This is a simple demonstration of how totally-unimodular constraint matrix ensures integral solution for linear program.

### 3.2.3 Totally-Unimodular Constraint Matrix

The particular constraint matrix for the linear program representing multiple object tracking problem represent the constraints of the system. It is straightforward to arrange column of the matrix in ascending order of time. Now each column represent one location $j$ at one particular time instance $t$. The constraint of the system can be classified into two sets i.e.

$$U1\colon\ \forall\, t, i\ \{\sum_{j \epsilon\, N(i)} f_{i,j}^{t}\ \ \leq 1\}$$

$$U2\colon\quad \forall t, i\ \left\{\sum_{j \epsilon\, N(i)} f_{i,j}^{t}\ -\ \sum_{k : i \epsilon\, N(k)} f_{k,i}^{t-1}\ \ \leq\ 0\right\}$$

$$\sum_{j \in N(v_{source})} f_{v_{source},j}\ -\ \sum_{k : v_{source} \in N(k)} f_{k,v_{sink}}\ \leq\ 0$$

The nature of these constraints makes it evident that for any particular column of the constraint matrix, there can be no more than three rows with non-zero entries and these entries belong to set $\{-1,0,1\}$.

There can be no more than one object reaching at a particular location $j$ from all of its neighbors at one time instance (U1). Difference between number of objects reaching at a location $j$ at one time instance from its neighboring location and number of objects reaching its neighboring locations a time instance earlier is either $0$ (if the object moved to the location $j$ ) or $-1$ (if the object did not move to $j$ ). The same property applies to *source* and *sink* nodes as neighbors for a particular location $j$ (U2).

The rows of the constraint matrix $C$ can be partitioned into two classes $U1\ and\ U2$ based on which constraints they specify. From theorem 1, we have established that $C$ is totally unimodular iff there exists a partition $\mathrm{R1}$ and $\mathrm{R2}$ for any of its sub-matrix $\mathrm{R}$ such that $\mathrm{R} = \mathrm{R1} \cup \mathrm{R2}$ and $\mathrm{R1} \cap \mathrm{R2} = \emptyset$ and

$$\forall j = 1,2,3 \ldots n \sum_{i \epsilon R1} a_{ij}\ -\ \sum_{i \in R2} a_{ij}\ \in\ \{-1,0,1\}$$

For the specific constraint matrix of the multiple object tracking problem, we assume that required partition is obtained as following

$$R1 = R \cap U1\ , R2 = R \cap U2$$

We know that for a particular column, there can be only three non-zero entries. For the partitions created by above formula, we can see that $R1$ can have only one non-zero entry and the value of entry can be 1. $R2$ Can have either $-1$ or $1$ if we move a row from $R1$ to$R2$. There a total of eight different possible cases.

| $\{a_{ij} \in R1\}$ | $\{a_{ij} \in R2\}$ | $\sum_{i\epsilon R1} a_{ij} - \sum_{i\in R2} a_{ij}$ |
|---|---|---|
| *{0…0,1}* | *{0…0}* | *1* |
| *{0…0,1}* | *{0…0,1}* | *0* |
| *{0…0,1}* | *{0…0,-1}* | *2* |
| *{0…0,1}* | *{0…0,1,-1}* | *1* |
| *{0…0}* | *{0…0}* | *0* |
| *{0…0}* | *{0…0,1}* | *-1* |
| *{0…0}* | *{0…0,-1}* | *1* |
| *{0…0}* | *{0…0,1,-1}* | *0* |

Only row 3 of the above table present a case where the property of total-unimodularity seems to be violated. But for this particular case, we can move the non-zero row from $R1$ to $R2$. Now the partition will correspond to case in row 8 which adheres to the property of total-unimodularity.

It can be concluded that there always exists a partition for the sub-matrix $R$ which satisfies theorem 1. Thus, constraint matrix is totally-unimodular and it ensures integral solution of the system even when the 'integer' condition has been relaxed.

# 4 K-path Optimization

Although solution to the continuous LP given above can be found, on average, in polynomial time but practical requirements, e.g. real time performance can't be met due to the large size of the system.

To be able to get multiple object tracking in reasonable time, the paper reformulates the ILP as KSP problem as discussed in the following sections [2].

## 4.1 KSP Problem:

Given a graph G(V,E), the goal is to find 'k' different shortest paths between a designated pair of nodes ($v_{source}$,$v_{sink}$). The problem has been studied extensively in the network optimization literature and

for different versions of the problem; solutions of varying degree of complexity (from NP-Complete to polynomial) exist.

The version of problem that was discussed in the paper that is main focus of this report deals with directed acyclic graphs (DAGs) with two specific constraints, viz. the nodes in the paths being disjoint and simple [2]. Node disjoint implies that no nodes are shared among different paths. Node simple means that loops are not allowed in the solution.

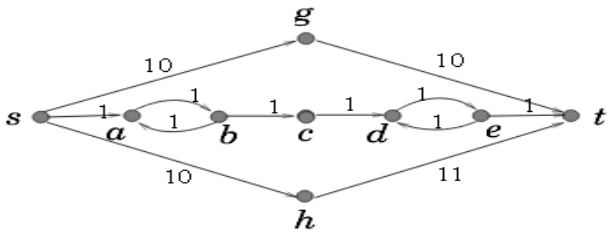


*Figure 4.The difference between simple and non-simple KSP: the three shortest simple paths have lengths 6,20,21. Without the simplicity constraint, paths may use the cycles (a,b,a) and (d,e,d), giving shortest paths of lengths 6,8,10 [7].*

Moreover, to track multiple objects, 'k' is discovered iteratively that minimize the overall cost (min-cost/max flow problem). The aforementioned constraints (node-disjoint and node-simple) apply naturally to the problem because two distinct objects do not share the same position simultaneously and the object trajectory is essentially loop-less with respect to time.

## 4.2 The KSP formulation of ILP:

The feasible solutions to ILP contain all solutions of KSP with arbitrary number of 'k' shortest paths. We, therefore, need only define the edge cost function in terms of the marginal posterior probability of an object at location 'i' given a video frame $I^t$ at time instant 't'.

$$c\left(e_{i,j}^{t}\right) = -log\left(\frac{\rho^{i}}{1-\rho^{i}}\right)$$

The edge cost function shows that edge cost can be negative if the probability estimate is low. The objective function for KSP is, therefore, the negation of that for ILP and is given as follows:

$$f^{*} = argmin_{f\in\mathfrak{H}} \sum_{t,i} c(e_{i,j}^{t}) \sum_{j\epsilon\, N(i)} f_{i,j}^{t}$$

The optimum value of 'k' for KSP is the one that minimizes the cost of flow (min. cost flow) and hence is the one that achieves maximum flow in the original ILP.

## 4.3 Finding k-shortest paths:

The paper implements an iterative algorithm of Suurballe [8] that uses Dijkstra's algorithm at every iteration for 'k',i.e., at most O(n) calls. The algorithm involves transformations to the original graph at every step to find alternate shortest routes (called as interlacings).

Given a directed graph G=(V,E), where 'V' is the set of vertices, and 'E' is the set of edges, the algorithm below computes the k-shortest paths between $v_{source}$ and $v_{sink}$ iteratively for l=1,2,…,k, where for each iteration 'k' is fixed. Thus at the $i^{th}$ iteration, l-shortest paths are computed by using *l-1* shortest paths from the previous iteration.

**Algorithm 1:** K-shortest paths algorithm for the tracking problem

**input** : a set of probabilistic occupancy maps
**output**: a set of $k$ paths between $v_{\text{source}}$ and $v_{\text{sink}}$

*Construct the initial graph* $G$
$p_1^* \leftarrow$ `generic_shortest_path` $(G, v_{\text{source}}, v_{\text{sink}})$
$P_1 \leftarrow \{p_1^*\}$
**for** $l \leftarrow 1$ **to** $l_{max}$ **do**
  **if** $l \neq 1$ **then**
    **if** $\text{cost}(P_l) \geq \text{cost}(P_{l-1})$ **then**
      **return** $P_{l-1} = \{p_1^*, \ldots, p_{l-1}^*\}$
    **end**
  **end**
  $G_l \leftarrow$ `extend_graph` $(G)$
  $G_l^c \leftarrow$ `transform_edge_cost` $(G_l)$
  $p_{l+1}^* \leftarrow$ `efficient_shortest_path` $(G_l^c, v_{\text{source}}, v_{\text{sink}})$
  $p^* \leftarrow$ `interlacing` $(P_l)$ `/* corres. to` $p_{l+1}^*$ `*/`
  $P_{l+1} \leftarrow P_l \cup p^*$ `/* i.e. augm. of` $P_l$ `and` $p^*$ `*/`
**end**

### 4.3.1 Explanation of the k-shortest path algorithm

Given the original flow graph, step-2 simply applies Dijkstra's single source shortest path algorithm to compute the shortest path tree which can be obtained in O(n) using topological sort of its vertices (the graph is a DAG!).

Step-6 simply checks if we have reached the optimal solution by comparing total cost at iteration '*l*' to that in the previous iteration '*l+1*'. If it is so, the set of all previous paths is returned and the algorithm exits. This is further explained in section 4.4.

Steps 10-11 involve transformations to the original graph in order to efficiently compute alternate non-decreasing cost paths as follows:

i) Every node in the previously computed shortest path is split into two nodes(except for the source and sink) respecting the in and out degree of the original node and then joining the split nodes with an auxiliary edge of zero cost.

ii) Every edge in the previous shortest path is reversed in direction (changing algebraic sign of the associated cost) including the auxiliary edges.

iii) All edge costs in the previous shortest path are changed as follows:

$$c'_{i,j} = c_{i,j} + s_i + s_j \quad \forall e_{i,j} \in E_l \; and \; s_n \; is \; the \; cost \; from \; source \; node \; to \; vertex \; \; v_n$$

The process of transformations is depicted in the following illustration (Figure 5).

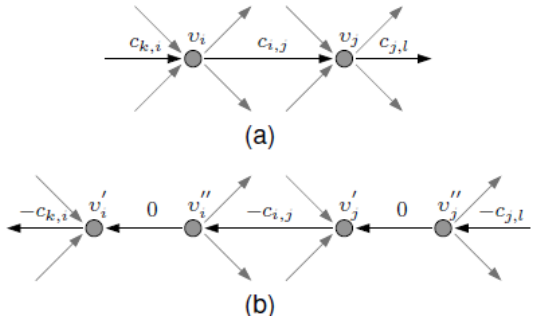


*Figure 5. (a) Original shortest path (shown in bold arrows) in the graph (b) Transformed path*

Step-12 is again a version of Dijkstra's algorithm specifically for non-negative edge costs. This step finds the edge-simple interlacing *p** of $P_l$.

Step-13 involves finding alternate shortest paths by the augmentation (referred to as interlacing in the paper) to the previously computed shortest path and the interlacing *p** from the previous step. It is important to note that shortest interlacings in the transformed graph (referred to as canonical graph) correspond one-to-one to that in original graph 'G'. The process of finding interlacing to $P_l$ using *p** is described in the following section.

Step-14 simply stores the shortest paths computed thus far.

### 4.3.2 Getting from $P_l$ to $P_{l+1}$:

Given that we have found the shortest path $P_l$ and interlacing *p** at iteration *'l'*, we can find the alternate path $P_{l+1}$at the next iteration *'l+1'* as:

i) Label each edge in '*p**' as '+' if it is in the direction from source to sink and '-' otherwise.
ii) An edge is common to both *p** and $P_l$ if and only if it has a negative label.
iii) A node is common to both *p** and $P_l$ if and only if it is incident to an edge with negative label.

Condition (ii) and (iii) add the constraint of the path being node-disjoint.

Finally $P_{l+1}$ is computed by adding positive labeled edges and removing negative labeled edges of *p** to $P_l$.

### 4.4 Guaranty of optimality of KSP solution:

At each iteration of finding l-shortest paths for l= 2,3,4…., we compute the total cost of the shortest paths as follows:

$$cost(P_l) = \sum_{i=1}^{l} cost(p_i^*)$$

In the above total cost function, each summand is given by the following:

$$cost(p_l^*) = \sum_{e_{i,j}^t \in p_l^*} c(e_{i,j}^t)$$

Since our path costs are monotonically increasing, i.e., $cost(p_{i+1}^*) \geq cost(p_i^*) \ \ \forall i$, the total cost function $cost(P_l)$ is convex with respect to $'l'$. Therefore, the global minimum is guaranteed when $cost(p_i^*)$ changes sign and becomes non-negative:

$$cost(P_{k-1}^*) \geq cost(P_k^*) \leq cost(P_{k+1}^*)$$

The above condition is set as the termination criterion in the algorithm.

## 5 Applications

The research described in this report has a number of contributions, which can be further applied to other problem. We will discuss some potential applications in the following section.

### 5.1 Optimization By Relaxing ILP To LP

Relaxation of integer linear program to linear program generally produces results different from optimal results of integer linear program, unless the constraint structure has the property of total-unimodularity. Classical examples are fractional vertex-cover and fractional graph-coloring. But important point is that every solution of integer linear program is also a solution of linear program. Thus quality of linear program solution is at least as good as that of integer linear program. For maximization

problem, linear program will produce a value equal to or larger than the solution of integer linear program. In minimization problems, linear program solution is equal to or smaller than the solution of integer linear program. Thus, this relaxation is good to finding upper or lower bound of the solution i.e. finding bounds for optimization problems.

**Branch and Bound approach:** One approach of [potentially reaching an integer solution while solving a continuous LP is to use *branch and bound* (B&B) algorithm. This is basically a divide-and-conquer approach in which problem is recursively split into disjoint sub-parts and only subparts holding potentially optimal values are kept while others are pruned. For example, if $x^*$ is the current optimal value in the solution of LP, LP is split into 2 parts; one with inequality $x \leq \lfloor x^* \rfloor$ and other with inequality $x \geq \lceil x^* \rceil$. Successive partitioning leads to optimal solution with integer values [5].

## 5.2 Unimodularity In 3D Segmentation

Shortest path algorithms have been used widely in image segmentation in 2D problems. In 3D segmentation, however, object boundary is actually a 2D surface. Thus, it is necessary to extend shortest path approach to minimal weight surface for 3D segmentation application. The new problem can be modeled as an instance of integer linear program.

$$\min_{z} Q(z) = \sum_{i} w_i z_i \,, \quad \textit{Subject to} \quad Bz = r$$

Where $z_i$ is the indicator variable indicating whether or not i$^{th}$ face is part of minimum weight surface or not, while $w_i$ is the weight of the face. Entries of constraint matrix indicate if the edge borders the face with coherent or anti-coherent orientation. $r_i$ Indicates coherence or anti-coherence of contour of i$^{th}$ edge. Constraint matrix is a totally-unimodular constraint matrix, making it possible for the problem to be solved by LP solvers [6].

## 5.3 Real Time Multiple Object Tracking

This work discusses how visually similar objects can be tracked with an algorithm with practically reasonable computational time. The reduction in complexity is important if this problem is to be solved in real-time. Thus, this work lays the ground work for real-time multiple visually-similar object tracking which has wide application in surveillance field. In a scenario like an airport or a train-station, surveillance videos have a huge number of humans. Tracking multiple humans in such videos is similar to tracking visually similar balls in test videos.

# 6 References:


1. Jiang, Hao, Sidney Fels, and James J. Little. "A linear programming approach for multiple object tracking." *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007.
2. Berclaz, Jerome, et al. "Multiple object tracking using k-shortest paths optimization." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*33.9 (2011): 1806-1819.
3. Robert J. Vanderbei. “Linear Programming: Foundation and Extensions” 2nd edition, Kluwer Academic Publishers.
4. Alexander Schrijver. “Theory of Integer and Linear Programming”, A Wiley-Interscience Publication
5. Raidl, Günther R., and Jakob Puchinger. "Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization."*Hybrid Metaheuristics*. Springer Berlin Heidelberg, 2008. 31-62.
6. Grady, Leo. "Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3D with application to segmentation." *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2006.
7. Hershberger J, Maxel M, Suri S (2007) Finding the k shortest simple paths: a new algorithm and its implementation. ACM Trans Algorithms 3:45
8. J. W. Suurballe, “Disjoint Paths in a Network,” Networks, vol. 4,pp. 125–145, 1974.