Identity Management as a Graph Partitioning Problem

Trevor Terris and Carlo Tomasi

March 28, 2012

Abstract

Identity management is the automatic inference of the identities of targets such as people or vehicles that are tracked by surveillance systems with many sensors. Measurement inaccuracies and ambiguities make identity management difficult, especially when targets remain invisible to the system for long periods of time as they move between sensors.

We cast identity management as the problem of partitioning a graph with both positive and negative edge weights. Each observation corresponds to a vertex, and edge weights encode the evidence for or against the hypothesis that the two observations at the edge's endpoint correspond to the same target. The resulting partition subdivides the graph into observations corresponding to separate objects.

This problem is translated into an Integer Program, proven to be NP-complete, and then addressed with a graph shrinkage and expansion technique. The small-graph version of the problem, obtained after shrinkage, is initialized with an algorithm that computes a set of observation pairs that are provably connected in the optimal solution.

Anecdotal experiments on simulations show the effectiveness of the approach.

1 Introduction

Identity management is the automatic inference of the identities of targets such as people or vehicles that are tracked by surveillance systems covering wide areas such as a shopping mall, a large harbor, or a multi-store building. Inputs for this task are the noisy, uncertain measurements made by sensors placed at strategic locations throughout the environment to be monitored. Even when sensors of the same type are used, different targets often produce similar measurements and, conversely, the same target can generate different measurements under different circumstances. In addition, different types of sensors may be called for in different locations, making direct comparison of sensor outputs problematic. These difficulties make identity management a challenge, especially when targets remain invisible to the system for long periods of time as they move between sensors.

We capture the information contained in a set of n measurements through m *association* values $0 \leq w_{ij} \leq 1$, defined as evidence for or against the hypothesis that measurements i and j were generated by the same target, given the values of the measurements. This choice contrasts with the more usual approach of summarizing measurements through n feature vectors, one per measurement, and then defining a metric in their space.

Pairwise associations are preferred over individual feature vectors for two reasons. First, sensors used in different parts of the space under observation can be of different types. The corresponding output values are then heterogeneous, and no single space is likely to fit both types of outputs naturally. Second, the time elapsed between the two measurements is an important source of information for computing the association value between them. This computation must consider the distance between the sensors, estimates of travel speeds, and the presence of possible delays (stores, restaurants, security lines, ...) or accelerators (moving walkways, escalators, ...) between the two measurement stations. These considerations cannot be captured by either measurement alone.

Because of these reasons, associations are more flexible and potentially richer than separate measurement features. Of course, associations can be computed from metrics defined in a feature space whenever the situation warrants – that is, when measurements happen to be homogeneous. Thus, associations subsume the standard approach, and provide a representational foundation for a broader set of circumstances.

In the following, we cast identity management as the problem of partitioning the graph G of observations with weights w_{ij} on itse edges. Following recent literature [3], we then translate that problem into an integer program. In Appendix A, we show that this problem is NP-complete with a proof that is different from that given in [2]. While Demaine *et al.* [3] solve the integer program by relaxation, we propose to contract edges in the graph that are likely to remain uncut in the optimal partition $\mathcal{S}$. Repeatedly applying these contractions shrinks the original graph G to a graph G' that is much smaller, as suggested in [5]. Once G' has been partitioned, the contracted edges can be re-expanded, and the resulting partition refined by simple, greedy optimization methods. To make contraction more effective, we introduce sufficient conditions that can be checked in an algorithmically efficient way for identifying edges that are *certain* to remain uncut in the optimal partition of G. These edges can be safely contracted and need not be re-expanded. In addition, edges that satisfy the sufficient condition provide initial constraints that can be added to the integer programming problem identified earlier, and the solution can be found more efficiently. We show the effectiveness of our approach through experiments on simulations.

2 Graph Partitioning

The measurements from the cameras can be represented as the set V of vertices in a weighted *association graph* $G = (V, E, W)$. Two measurements i and j in V are connected by an edge in $e_{ij} \in E$ if an association value $w_{ij} \in W$ is available for them, and w_{ij} is the *weight* for that edge.[1] The graph G is generally not complete, as it does not always make sense to establish associations for two measurements. For instance, two sensors may be cameras that look at people from different directions (perhaps from the front and from the back), or across excessively long time intervals, and this may make matching between these views meaningless. Appendix B shows properties that the weights w_{ij} are to satisfy if they are interpreted as measures of the evidence for or against the hypothesis that vertices i and j correspond to the same target. These properties are not required to hold for the algorithms given below.

The *identity management* problem then amounts to determining a clustering of G into sets that correspond to different targets and that agrees as much as possible with the given edge weights in W. A clustering of G is a partition

$$\mathcal{S} = \{S_1, \ldots, S_k\}$$

of the set V of observations:

$$S_1 \cup \ldots \cup S_k = V \quad \text{and} \quad i \neq j \Rightarrow S_i \cap S_j = \emptyset \; .$$

The sets S_i are called *clusters*.

To define agreement with the edge weights, let $C(v)$ be the set of vertices $v \in V$ that are in the same cluster as v. Furthermore, let E^+ and E^- be the sets of edges with positive and negative weights respectively (zero-weight edges are removed). Then, the weight of a clustering $\mathcal{S}$ is defined as follows:

$$w(\mathcal{S}) = \sum_{(u,v)\in E^+ \mid u \notin C(v)} w(u,v) - \sum_{(u,v)\in E^- \mid u \in C(v)} w(u,v) \; .$$

In words, the weight of a clustering is the sum of the absolute values of those edges that disagree with the clustering: Edges with positive weight disagree when they

[1]For convenience, the reflexive property is ignored throughout this paper. In other words, a measurement is not associated with, or considered equivalent to itself.

connect vertices in different clusters; edges with negative weights disagree when they connect vertices in the same cluster.

Identity management then determines

$$\hat{\mathcal{S}} = \arg\min_{\mathcal{S}} \; w(\mathcal{S})$$

over all possible partitions $\mathcal{S}$ of V. We call the optimal value $w(\hat{\mathcal{S}})$ of $w(\mathcal{S})$ the *best partition value* of G, and $\hat{\mathcal{S}}$ the *best partition* of G.

This optimization problem is similar to the graph-cut graph problem considered by Kernighan and Lin [6] and Andreev and Räcke [1]. However, instead of dividing the graph into a predefined number of clusters, the graph is partitioned into however many clusters are required to minimize the sum of the weights of edges between vertices in different sets. In other words, the number k of clusters is an output of the algorithm, rather than an input.

3 Graph Partitioning as an Integer Program

Following recent literature [3], we rephrase this problem as an integer program. Let $x(u, v) = x(v, u)$ be 0 if u and v are in the same cluster and 1 otherwise. Then,

$$w(\mathcal{S}) = \sum_{(u,v)\in E^+} w(u, v)\,(1 - x(u, v)) - \sum_{(u,v)\in E^-} w(u, v)\,x(u, v)\ .$$

Identity management then seeks a valid assignment of each $x(u, v)$ in $\{0, 1\}$ to minimize this cost. An assignment is *valid* if the $x(u, v)$ satisfy the triangle inequality. To see this, note that $x(u, v) = 0$ flags vertices in the same cluster. Set membership is transitive, so if a triangle has two zero weights – say, $x(a, b) = x(b, c) = 0$, then $x(a, c) = 0$ by transitivity, so that the third edge must be zero as well. By listing the four possible combinations of zeros and ones on the edges of a triangle (0, 1, 2, or 3 zeros), it is easy to see that for valid configurations

$$x(a, b) + x(b, c) \geq x(a, c)\ .$$

In summary, identity management is the solution of the following integer program:

$$\min \sum_{(u,v)\in E^+} w(u, v)\,(1 - x(u, v)) - \sum_{(u,v)\in E^-} w(u, v)\,x(u, v) \qquad (1)$$

subject to

$$\begin{aligned}
&x(u,v) \in \{0,1\} \\
&x(a,b) + x(b,c) \geq x(a,c) \\
&x(u,v) = x(v,u) \ .
\end{aligned}$$

If this integer program is relaxed into a linear program – that is, if fractional assignments $x(u,v) \in [0,1]$ are allowed – the integrality gap is shown [3] to be $\Omega(\log n)$ where n is the number of vertices in V, so that rounding the LP solution into an integral one gives a $O(\log n)$ approximation factor. An algorithm to achieve this approximation was proposed by Demaine *et al.* [3].

4 Crossing Sets

Reliance on linear programming and rounding guarantees polynomial time, but actual running times are still slow, and the $O(\log n)$ approximation factor may not be sufficient for identity management. Instead, we propose to contract edges in E that are likely to remain uncut in the optimal partition $\mathcal{S}$. Repeatedly applying these contractions shrinks the original graph G to a graph G' that is much smaller, as suggested in [5]. Once G' has been partitioned, the contracted edges can be re-expanded, and the resulting partition refined by simple, greedy optimization methods.

To initialize the solution of the partition problem for the shrunk graph G', we define sufficient conditions that can be checked in an algorithmically efficient way for identifying edges in E that are *certain* to remain uncut in the optimal partition $\mathcal{S}$ of G'. These edges can be safely contracted and need not be re-expanded. In addition, edges that satisfy the sufficient condition determine subsets of the sets $S_1, \ldots, S_k$ in $\mathcal{S}$. These subsets provide initial constraints that can be added to the integer programming problem identified earlier, and the solution can be found more efficiently.

More specifically, we propose in Section 6 a polynomial-time algorithm that takes a graph $G\,(V, E, W)$ and returns a (possibly empty) set of edges such that if $e_{ij} \in E$ is returned, then vertices i and $j \in V$ will be grouped together in the solution that produces the best partition value. This algorithm is centered on the notion of a *crossing set* for an edge e_{ij} with endpoint vertices i and j, which we explain next.

Consider an edge $e_{ij} \in E$. Imagine laying out all vertices of V in a row, with the only requirement that vertices i and j be next to each other. Now draw in the edges; edges connecting adjacent vertices, like i and j, are short and horizontal, but edges that cross multiple vertices to connect two far apart vertices must arc over some part of the graph, as shown in Figure 1.

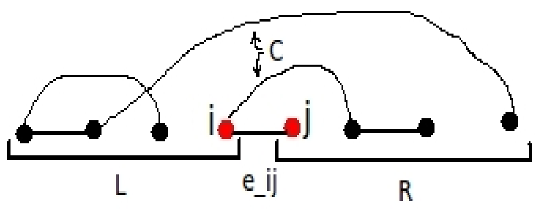


Figure 1: Vertices of the graph

Let us divide V into two groups of vertices, L and R, where L is the set of vertices including i that are to the left of e_{ij}, and R is the set of vertices including j that are to the right of e_{ij}. Then, place all edges except for e_{ij} that connect a vertex from L to a vertex in R into a set C. C is said to be a *crossing set* for e_{ij} in graph G. Then, we show in Appendix C that the following results holds.

Theorem 4.1 *Given a weighted graph $G = (V, E, W)$ with edge weights $w(e)$, let C be a crossing set for any given edge e_{ij}. Then, the endpoints i and j are in the same set of the best partition $\mathcal{S}$ of G if*

$$w(e_{ij}) > \sum_{e \in C} |w(e)| \ . \tag{2}$$

There are many possible crossing sets for any given edge e_{ij}. We have developed heuristics aimed at choosing a crossing set that is as useful as possible – that is, one for which the condition (2) is as loose as possible – so that many vertex pairs will be assigned to sets of $\mathcal{S}$, thereby providing a good initial partition for subsequent refinement. This line of reasoning leads to the algorithm sketched in pseudocode in the next Section.

5 Heuristics

Any linear ordering of the vertices in V that leaves v_i and v_j adjacent to each other leads to a crossing set, with its attending predicate. However, not all crossing sets lead to equally useful predicates. We now introduces heuristic considerations diesigned to loosen the bound in the predicate as much as possible, so that more edges will be included in the final answer.

5.1 Using Prohibitively Negative Edges and Matching

We can decrease the value that the weight of e_{ij} has to be greater than, and thus increase the usefulness of the algorithm, by taking advantage of the existence of prohibitively negative edges. These edges are common in the identity management problem, because observations that are in the same frame cannot correspond to the same person. Vertices corresponding to such observations would therefore be connected by edges with very large negative weights.

5.1.1 Reasoning

Consider the value

$$\hat{w} = \sum_{e \in E} -\max(w_e, 0) \ .$$

If an edge e_{ij} has a weight less than $\hat{w}$, then a potential solution that places vertices i and j together cannot be the solution that produces the best partition value. This is because the trivial solution of placing each vertex in a different set will produce a larger value than any solution that groups i and j together.

We then create a set S, such that $S_k \in S$ is a set of vertices from V. We assign vertices to the sets in S such that $\forall m, n \in S_k, \mathrm{w}\left(e_{mn}\right) < \hat{w}$. We can call sets of vertices with this property "incompatible cliques." This can be found easily in the identity management problem, by just placing a set of vertices in a set in S if the vertices all correspond to observations from the same frame.

Thus, we know that if we group vertices together to get the best partition value, the vertices grouped together between any two sets $S_i, S_j \in S$ will form a (possibly incomplete) bipartite matching between S_i and S_j. We can now use crossing edges, with some modifications. We create a smaller graph, $G''(V'', E'', W'')$, such that for every $S_k \in S$, we create a vertex $k'' \in V''$. If vertex k'' is in R and l'' is in L, then C will include all edges e_{mn} where m is an element of S_k and n is an element of S_l. Thus, when we find the grouped graph G'', V'' is a set of vertices, and E'' is a set of edges such that

$$\mathrm{w}(e''_{kl}) = \sum_{m \in S_k} \sum_{n \in S_l} |\mathrm{w}(e_{mn})| \ .$$

We can then find the sum of the edges in C.

Unlike the original problem, the penalty from placing i and j in different partitions is not simply the weight of e_{ij}: if e_{ij} is in the optimal local solution, and we cut it, the new best solution might be a matching which does not include e_{ij} but has only a slightly smaller penalty.

To calculate the penalty from cutting an edge, we use the Hungarian algorithm for weighted bipartite graph matching [7]. We find the best matching, then for each edge in the solution, we set the weight to zero, and find a new matching; the difference between the solutions is the penalty from cutting the edge. If this value exceeds the sum of the weights of the crossing edges, then the edge is necessarily in the final graph.

5.1.2 Algorithm

Using $\hat{w}$, we assign vertices to groups $S_{0,\ldots,n} \in S$ such that $\forall v_i, v_j \in S_k, \mathrm{w}\left(e_{ij}\right) < \hat{w}$. This can be found by placing all vertices corresponding to observations from the same frame into the same set in S.

Then, we create a graph $G''\left(V'', E'', W''\right)$ with each group S_k corresponding to a vertex in V'', and the weights of the edges of G'' being the sum of the absolute values of the edges between the vertices in the groups. That is, we create a graph $G''(V'', E'', W'')$ such that $\forall S_k \in S,$ we add a new $k'' \in V''$, and $\forall k'', l'' \in V''$,

$$\mathrm{w}\left(e''_{kl}\right) = \sum_{i \in S_k} \sum_{j \in S_l} \left|\mathrm{w}\left(e_{ij}\right)\right|$$

Later, we find the maximum bipartite matching for the edges e_{ij} between all vertices $i \in S_k$ and $j \in S_l$, returning a set of edges M. Also, for each edge $e_{ij} \in M$, we temporarily set $\mathrm{w}\left(e_{ij}\right) = 0$, and find a new matching M'. If

$$\sum_{e \in M} \mathrm{w}\left(e\right) - \sum_{e \in C} \left|w_e\right| > \sum_{e \in M'} \mathrm{w}\left(e\right)$$

then that edge e_{ij} is included in the final graph.

5.2 Edge Weights in G''

When discussing G'', we mentioned that we could let the edge weight of e''_{kl} be equal to the sum of the absolute values of the weights of edges e_{ij} such that $i \in S_k$ and $j \in S_l$. However, we can find a smaller value for the edge weights in G''.

5.2.1 Reasoning

In Appendix C, we showed that an edge e_{ij} can be included with certainty if inequality 2 on page 6 is true. In the sum, we add together the weights of the edges that, in the worst case, will be preserved when e_{ij} is maintained and the edges that will be cut if e_{ij} is cut. We stated that in the worst case, every edge in C will incur a penalty.

However, if we have two vertices k'' and l'' in V'', then not all edges between S_k and S_l will be preserved, even in the worst case. This is because preserving every such edge might result in an edge within S_k or S_l being preserved. So, the most negative edges that might be preserved in the worst case would really be a matching of the negative edges from S_k to S_l, and the largest number of positive edges would be a matching of the positive edges from S_k to S_l.

To find a tighter bound, then, we calculate the best bipartite matching of positive edges between the nodes in S_k and S_l and the negative edges between S_k and S_l. We then set the weight of the edge between k'' and l'' equal to the sum of the absolute value of the weights of the edges in the two bipartite matchings.

5.2.2 Algorithm

First, we find the edges in maximum bipartite matching of the vertices in S_k and S_l; we can denote this set of edges M_{pos}. Then, for each e_{ij} such that $i \in S_k$ and $j \in S_l$, we set $w_{ij} = -w_{ij}$, and find a new matching, producing a set of edges that we can call M_{neg}.

We then let w''_{jk} be the sum:

$$w''_{jk} = \sum_{e \in M_{pos}} max(w_e, 0) - \sum_{e \in M_{neg}} min(w_e, 0)$$

So, we find the maximum positive and negative matching and add their absolute values together, in order to obtain w''_{jk}.

5.3 Using Max Flow

This section is designed to further reduce the value that w_{ij} has to exceed, by finding the best arrangement of vertices into L and R.

5.3.1 Reasoning

Finding an arrangement of vertices on the line is nontrivial. For example, if vertices a and b are in the same frame in the identity management problem, then e_{ab} will be a very large negative edge. Then, if a is in R and b is in L when we consider a vertex e_{ij}, we see that e_{ij} will never be in the final answer, because the very large edge e_{ab} will be in C, dominating the value of e_{ij}.

Just a few poorly placed vertices, then, can result in every edge not being included, no matter how dominant, because of large crossing edges. Therefore, it would be helpful to efficiently group the vertices into L and R such that the sum of the weights of the edges in C would be minimized. This can be found using the max-flow min-cut theorem: if we find the maximum flow from one vertex to another, like vertices i and j, we could find the sum of the smallest cut that would separate the graph in two subgraphs, one with vertex i, the other with vertex j.

We find the maximum flow between vertices i and j, setting the flow capacity between any two vertices a and b to $|\mathrm{w}(e_{ab})|$. We can split the graph into two subgraphs. The vertices of the subgraph containing vertex i would be assigned to L, and the vertices of the one containing vertex j would be assigned to R, and the cut edges would be the edges in C and e_{ij}.

We do not need to actually find the groups, though; we just find the maximum flow between i and j, and subtract from that the weight of e_{ij} to obtain the sum of the weights of the edges in C. If we use the absolute values of the weights when we calculate flow, then the sum will be

$$\sum_{e_{mn} \in C} |\mathrm{w}(e_{mn})|$$

Which, as we found earlier, is the value that an edge e_{ij} has to exceed to be included in the final graph.

However, the only value we need from C is the sum of the weights, which is calculated by the flow. So, we can use this sum without ever knowing which edges are in C, or what vertices are in L and R.

5.3.2 Algorithm

We use a max-flow min-cut algorithm (specifically the Edmonds-Karp algorithm [4]) to find the maximum flow between every pair of vertices in V'', where $flow_{kl}$ is the amount of flow between vertices $k'', l'' \in V''$.

So, given a graph $G(V, E, W)$, we group it into a set of a set of vertices, S, and create a graph $G''(V'', E'', W'')$ where for each $S_k \in S$ we add a vertext k'' to V'', and the weight of the edge e'_{kl} is set to

$$w'_{kl} = \sum_{e \in M_{pos}} max(w_e, 0) - \sum_{e \in M_{neg}} min(w_e, 0)$$

We then find the max flow from one vertex k'' to another vertex l'', and subtract from this the weight of e'_{kl} to find the sum of the absolute values of the weights in C.

6 Initialization Algorithm

Algorithm 1 on page 12 summarizes the considerations in the previous Section, and finds an initial grouping of vertices in a graph G by using condition (2) to compute pairs of vertices that belong to the same set of the best-value partition of G.

Given graph $G = (V, E, W)$ with edge weights $w(e)$, the routine `collapse` first collapses all the sub-cliques of G into individual vertices $v'' \in V''$ whenever all edges in the clique have weight $\hat{w}$. The weight of the edge between vertices u'' and v'' in V'' is set to the sum of the two matchings between the edges in S_u and S_v, as described in section 5.2.

This step possibly reduces the size of G to produce a new graph $G'' = (V'', E'', W'')$. In the pseudocode below, the symbols u'' and v'' are also used interchangeably to denote vertices in V'' and the sets in V that vertices in V'' correspond to.

The algorithm then repeatedly calls a routine `maxMatching`(U, V) that returns the maximum matching of a bipartite graph with vertex sets U and V. The routine `maxFlow`(G, u, v) computes the maximum flow between vertices u and v in graph G.

The algorithm produces a set `keep` of edges to be kept, that is, of vertex pairs that belong to the same set in the best-value partition $\mathcal{S}$.

In Appendix D we prove the complexity of this algorithm to be $O(|V|^2(|V||E|^2 + |V_{max}|^4))$, where V_{max} is the maximum number of vertices in a set of S.

7 Experiments

To experiment with these ideas, we have developed software that simulates trajectories and observations. Our system generates random trajectories in a rectangle, and produces observation vectors that conform to a given observation noise model.

A screenshot of the simulation produced by this system is shown in Figure 2. The screenshot depicts, from left to right: the moving objects being simulated, with their ground truth IDs; the edges connecting the observations, with line width corresponding to edge weight; a display of the accuracy of the tracking algorithm, explained further in Section 7. In the second image, the edges are displayed only between observations that are one frame apart. In the third image, edges with weight less than or equal to zero are not displayed.

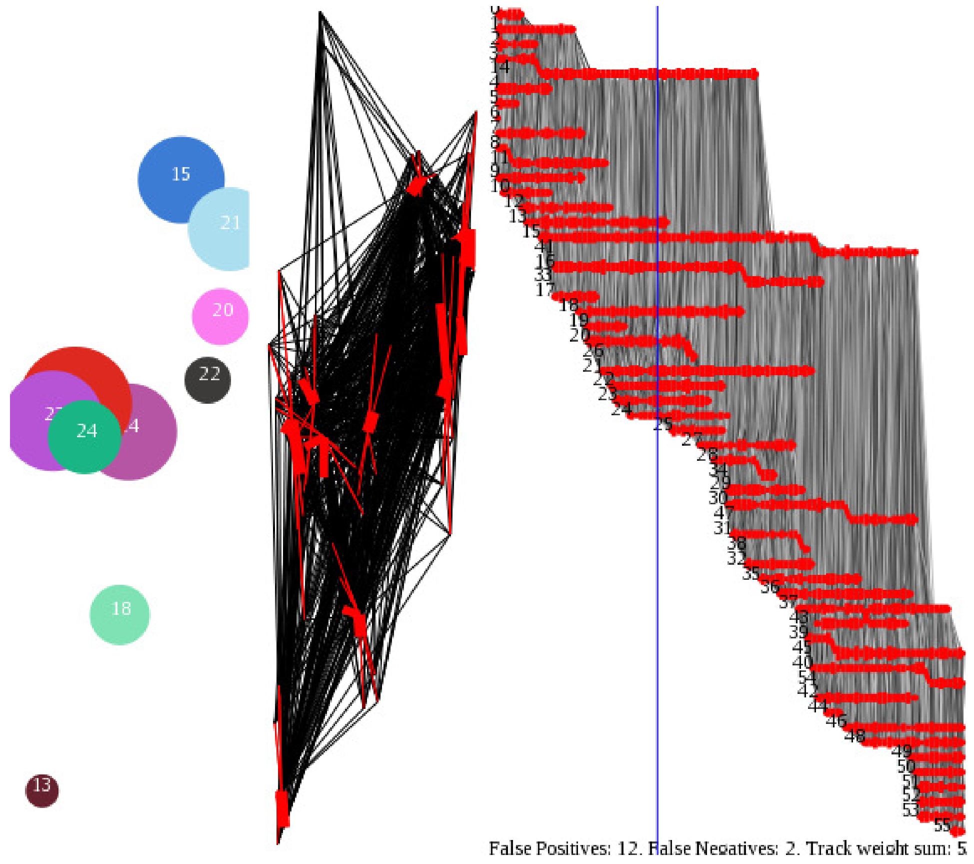
We calculated the edge weights based on the features of the observations. Let δx, δy, and δt be the differences between a pair of observations on the x-axis, y-axis, and time respectively, and δf_k be the difference between the observations along the kth feature. Then, the weight of the edge connecting the two observations is $e^P - D_1 - D_2$, with:

$$D_1 = max(5 \cdot (0.1 - d_1), 0.0)e^{2(\delta t - 1)} \ ,$$

$$D_2 = max(5 \cdot (0.1 - d_2), 0.0)e^{2(\delta t - 1)}$$

Algorithm 1 Initialization of the Best-Value Partition.

```
keep $= \emptyset$
$G = (V, E, W)$ // input graph
$G'' = (V'', E'', W'') =$ collapse$(G)$ // collapsed graph
**for** $e'' = (u'', v'') \in E''$ **do**
  $w(e'') = 0$
  $M^+ =$ maxMatching$(u'', v'')$
  **for** $e \in E$ **do**
    $w(e) = -w(e)$
  **end for**
  $M^- =$ maxMatching$(u'', v'')$
  **for** $e \in E$ **do**
    $w(e) = -w(e)$
  **end for**
  **for** $e \in M^+ \cup M^-$ **do**
    $w(e'') = w(e'') + |w(e)|$
  **end for**
**end for**
**for** $e'' = (u'', v'') \in E''$ **do**
  $\phi =$ maxFlow$(G'', u'', v'') - w(e'')$
  $E_{\text{max}} =$ maxMatching$(u'', v'')$
  **for** $e \in E_{\text{max}}$ **do**
    $w_{\text{orig}} = w(e)$
    $w(e) = 0$
    $E_{\text{new}} =$ maxMatching$(u'', v'')$
    **if** $\sum_{e \in E_{\text{max}}} w(e) - \sum_{e \in E_{\text{new}}} w(e) > \phi$ **then**
      keep $=$ keep $\cup \{e\}$
    **end if**
    $w(e) = w_{\text{orig}}$
  **end for**
**end for**
return keep
```



False Positives: 12. False Negatives: 2. Track weight sum: 5

Figure 2: A screenshot from our random-trajectory observation simulator.

where d_1 is the distance of the first observation and d_2 the distance of the second observation from the edge of the field of view, and

$$P = -\frac{5}{12}\sqrt{10\delta x^2 + 10\delta z^2} + 2(\delta t - 1) + 0.6\sum_k \delta f_k \ .$$

This distribution has the property that the edge weights tend toward zero as δt increases. Edge weights of observations near the edge of the field of view are penalized, to prevent an observation of an object leaving the field being grouped with an observation of an object entering.

The distribution of edge weights can be seen in Figures 3 and 4. In each graph, the x-axis is the weight of the edge, and the y-axis is the logarithm base 10 of the number of edges with that weight. Figure 3 is the log distribution of the edge weights of all edges in a simulation, while Figure 4 is the distribution of the weigths of edges connecting observations of the same object. These distributions are from 1000 observations, resulting in 499500 edges (a complete graph, except for edges e_{ij} where $i \geq j$).

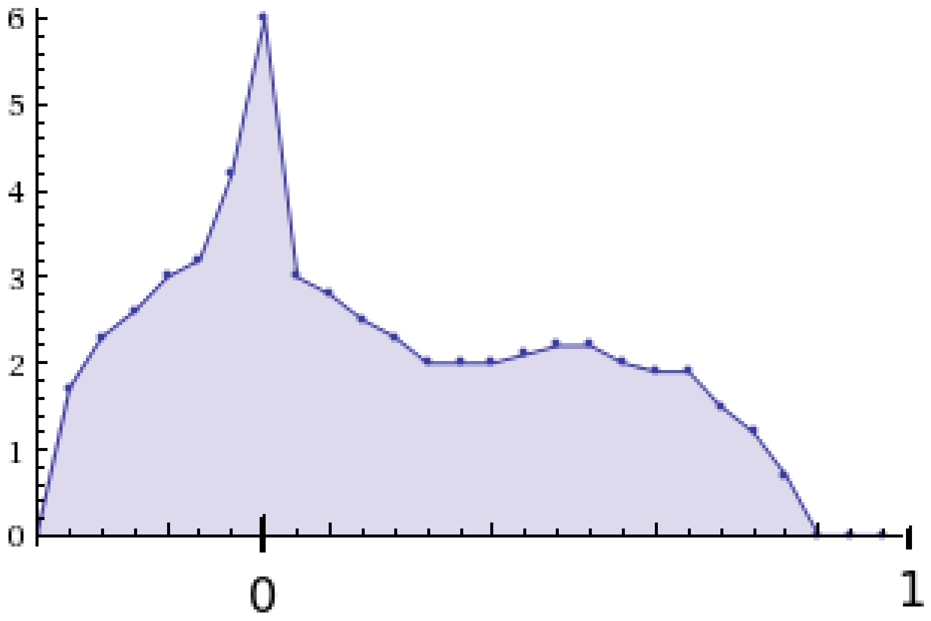
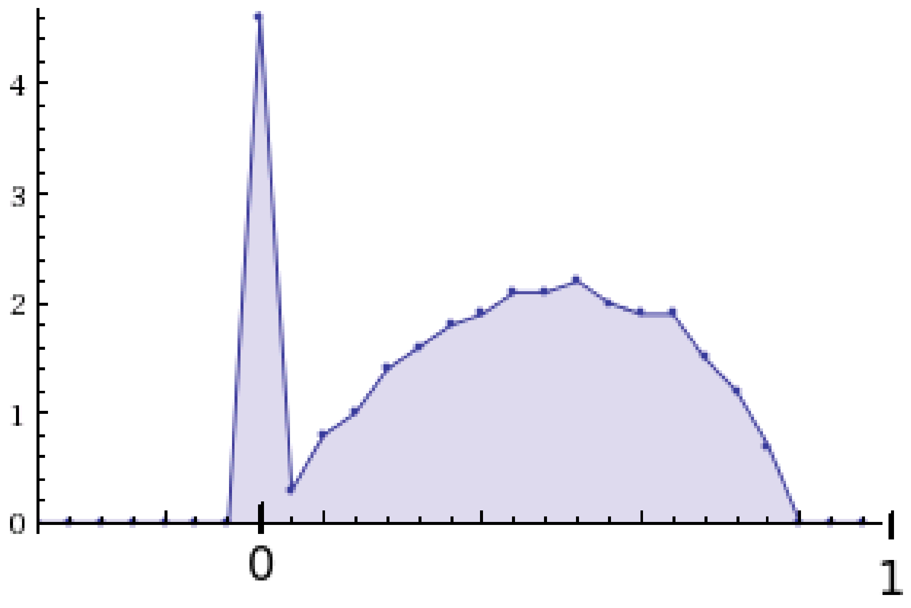


Figure 3: The distribution of all edge weights (negative infinity edge weights omitted).

Figures 5 and 6 show two sample outputs from Algorithm 1 in Section 6. In these diagrams, the vertices aligned on the same horizontal line correspond to ob-



,

Figure 4: The distribution of edge weights of edges connecting observations of the same object.

servations that, according to (known) reference truth, are of the same object. Red edges connect vertices that the algorithm groups together. Gray segments connect vertices for which positive weights are available, but that are not grouped by the algorithm. Their exact shade is randomized, so that different edges are easier to see. The width of both red and gray lines is roughly proportional to edge weight; negative-weight edges are not displayed.

Thus, long horizontal stretches of red indicate true positives, that is, observations of the same target that were correctly linked to each other by the initialization Algorithm 1 in Section 6. Horizontal gray line segments indicate false negatives, that is, edges that link two observations of the same target that were not grouped together. Slanted red line segments indicate false positives, that is, edges that link observations of different targets that were erroneously grouped together.

The graph in Figure 5 has only weights with magnitude greater than 0.3. Since uncertain observations (with small weights) are not present, inference is rather reliable. Row 2, for example, is a single solid section of horizontal red. That is, all the vertices of object 2 were grouped together, without missing an observation or including an observation of another object.

In the edges for the first object, a gap in the path is visible. This corresponds to over segmentation, with multiple clusters corresponding to observations of a single object.

When small weights are present, as in Figure 6, inference is more difficult, and a few errors occur. In particular, the problem of the edges that extend after the end of a track or before the beginning of one has become more frequent. That is because these edges are usually much smaller than the true edges; however, as the cutoff is decreased, they are included more and more often, and there is not a strong true edge that competes with them. While the tracking algorithm minimized the sum $w(\mathcal{S})$ correctly, the input edge weights did not constrain the desired solution uniquely.

Finally, in figure 7, nearly all edges were preserved from the original graph (cutoff is 10^{-7}), and we see the conclusion of this trend. In this example, there are no false negatives, but two false positives. These false positives, though, are instances where an edge is included from the end of one track to the beginning of another.

All of these examples so far were taken from a simulation with one simulated camera observing the objects. The next two examples are from simulations with two cameras. The fields of view for the cameras can be seen in figure 8.

In the first example, we alternate the frame numbers with the cameras. So, in the even-numbered frames, we have the observations from camera zero; in the odd-numbered frames, we have the observations from camera one. The cameras have a large amount of overlap in their fields of view, but there are sections of each

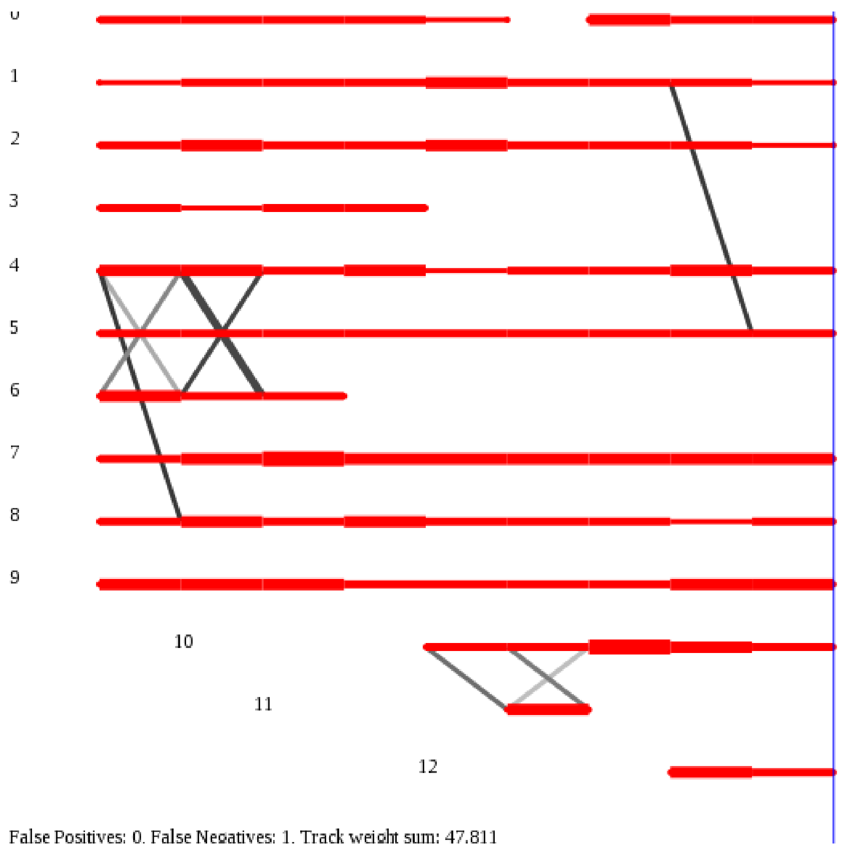


Figure 5: Graph with low-ambiguity observations, characterized by edge weights with magnitude greater than 0.3.

camera's field that the other cannot see. The results of the tracking can be see in figure 9. As one can see, the results are not nearly as accurate as they were in the earlier examples of one camera.

Examining the results, we see that most tracks have one section of accuracy, but other sections where no observations are grouped together. The section of accuracy corresponds to areas where the object was in the field of view of both cameras; after the objects entered the field of just one camera, the frame difference (two frames, rather than one) resulted in very reduced weights. These weights were low enough that they were not able to exceed the sum of the weights in C, so none of these weights were included.

In the final example, the cameras are arranged the same as before (large amounts of overlap, and of fields of view exclusive to one camera). The one difference is

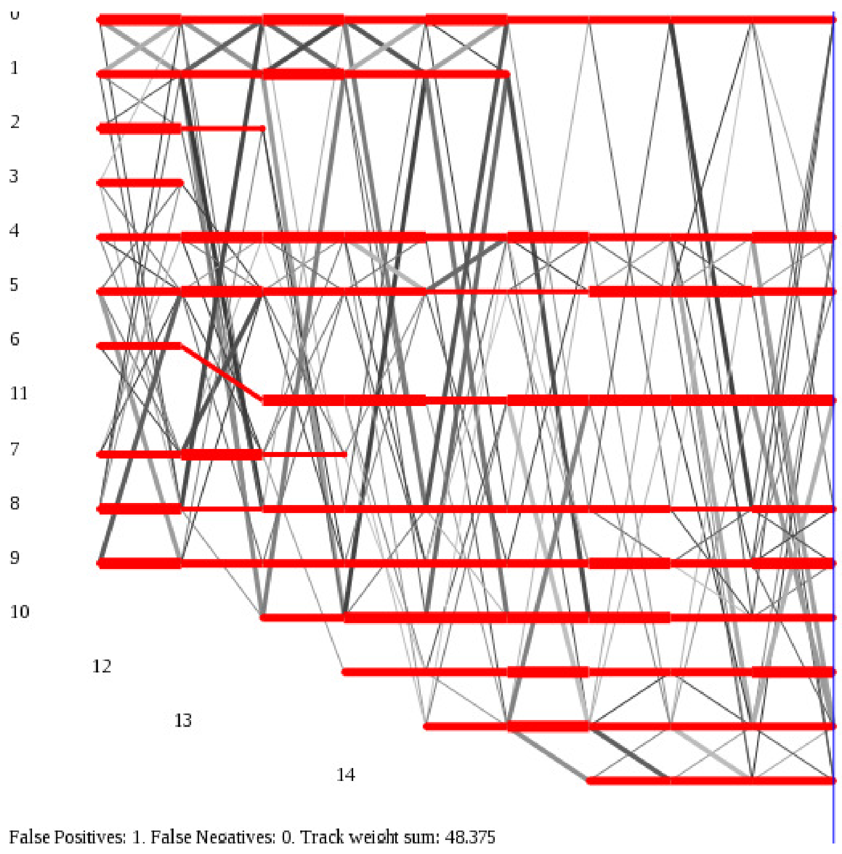

0
1
2
3
4
5
6
11
7
8
9
10
12
13
14
False Positives: 1. False Negatives: 0. Track weight sum: 48.375

Figure 6: Graph with more ambiguous observations than in Figure 5, containing small-magnitude edge weights as well.

that the observations in each camera, rather than alternating by frame, are in each frame. So, each frame contains observations from both cameras zero and one. We might think that this will remove the problem we had earlier, where edges connecting observations from the same camera had weights that were too small, because the frame difference was too large.

However, by examining figure 10, we see that this is not the case. In fact, not a single vertex is grouped with another vertex. The reason for this is that the edges in C are often just as large as the edge e_{ij} being considered. Because there are multiple edges in C, then, the sum of the weights in C is much larger than the weight of any single edge. So, no edge can be included with certainty.

For each of the five cases, we ran 100 trials, where each trial is an instance of the problem with 1000 vertices. We found the sum of the number of false positives,

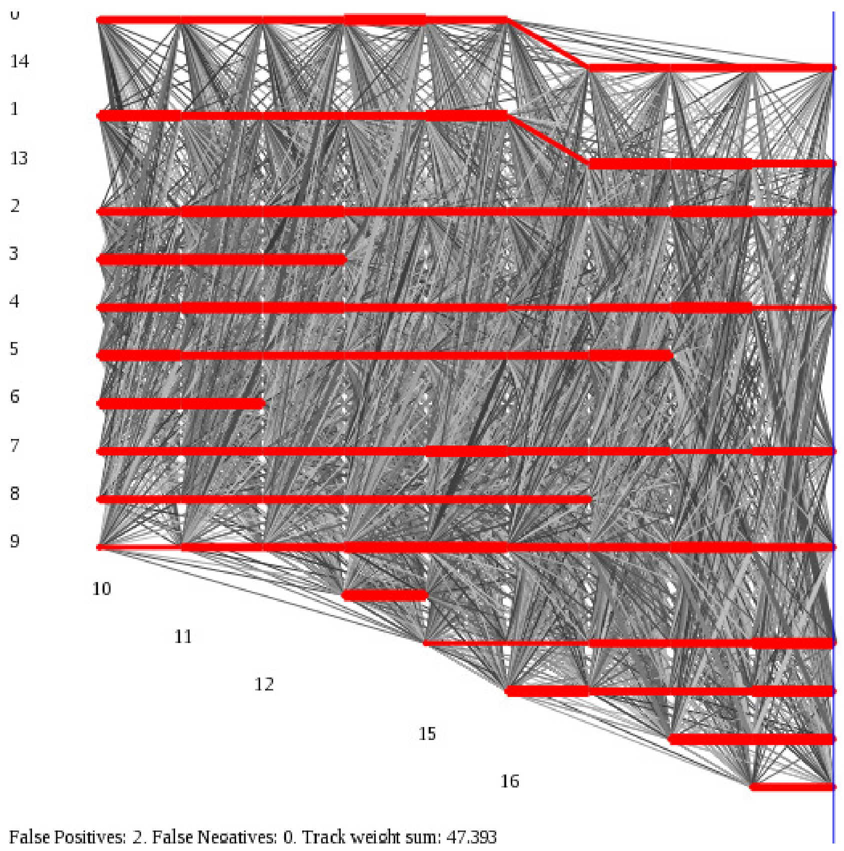

0
14
1
13
2
3
4
5
6
7
8
9
10
11
12
15
16
False Positives: 2. False Negatives: 0. Track weight sum: 47.393

Figure 7: Weight cutoff is 0.0000001

false negatives, and the total score found in each case.

For the weight cutoff of 0.3, there were a total of 426 false positives and 718 false negatives, with a total score of 52873.8. When the cutoff was 0.1, there were 1461 false positives, 351 false negatives, and a total score of 53318.3. when the cutoff was 10^{-7}, there were 1090 false positives, 352 false negatives, and a total score of 54051.0. Interestingly, the number of false positives is lower when the cutoff is 10^{-7}. This is probably because, with a lower cutoff, there a many more edges that are in C for a given edge. Therefore, even though the additional edges have very small weights, the extra edges exceed the value of edges when their values are very small, which means they probably aren't correct.

When the frames of the cameras did not overlap, there were 1788 false positives, 20655 false negatives, and a total score of 23387.2. Finally, when the frames did overlap, there was 1 false positive, 93265 false negatives, and the total score

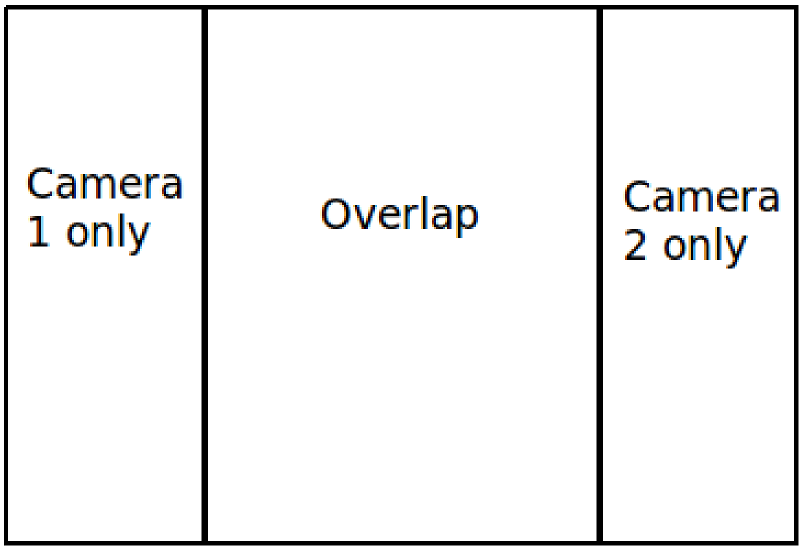
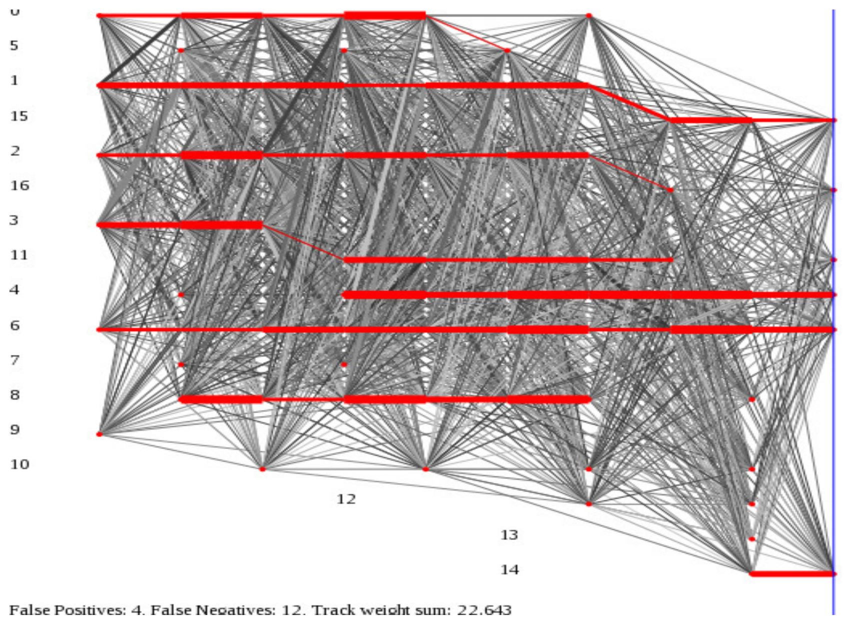


Figure 8: Areas covered by the cameras



False Positives: 4. False Negatives: 12. Track weight sum: 22.643

Figure 9: Two cameras, without overlapping observations in frames.

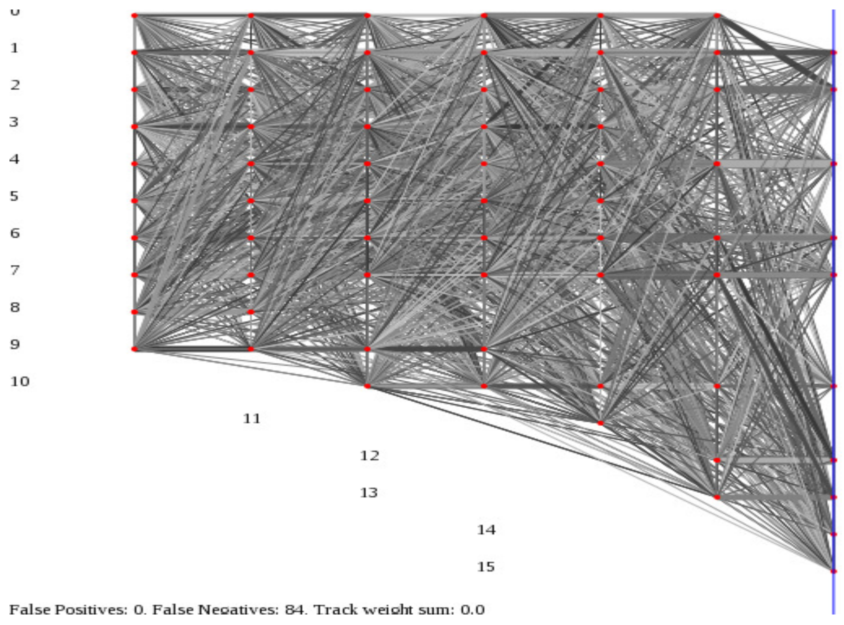

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
False Positives: 0. False Negatives: 84. Track weight sum: 0.0

Figure 10: Two cameras, with overlapping observations in frames.

was 15.5.

References

[1] Konstantin Andreev and Harald Rcke. Balanced graph partitioning. In *In 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 120–124. ACM Press, 2004.

[2] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *IEEE Symposium on the Foundations of Computer Science*, pages 238–250, 2002.

[3] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361:172–187, 2006.

[4] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[5] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientic Computing*, 20(1):359–392, 1999.

[6] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[7] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

A Graph Partitioning is NP-Complete

The graph partitioning problem defined in this paper is now shown to be NP-complete by reducing 3-SAT to it. To do this, we design a graph such that the best partition value is equal to a certain minimum amount only when the variables are consistent and all clauses are satisfied. That is, we create a graph such that parts of the graph correspond to the values of the boolean variables. The minimum penalty is obtained when each variable is one value, false or true, in every clause. If any clause is not satisfied, an infinite penalty is added.

So, if we create m instances of a subgraph for the variables, and the minimum penalty for each subgraph is n, then by finding the best partition value, we solve 3-SAT. If the penalty is mn, then there is a consistent set of variable assignments that results in the clauses being satisfied. If the penalty is greater than mn, then there is no such assignment; the only way to satisfy the clauses, and prevent the infinite penalty, is to use an inconsistent and more costly assignment of values.

Consider an instance of the 3-SAT problem. For each clause, create the graph in Figure 11.

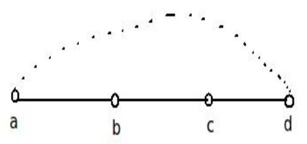


Figure 11: 3-SAT clause

Each boolean variable is assigned to an edge, and the dotted line represents an edge with a very large negative weight. Thus, if we have the clause $(x_1 \vee x_2 \vee \neg x_3)$, then x_1 would be assigned to edge e_{ab}, x_2 would be assigned to e_{bc}, and $\neg x_3$ would be assigned to e_{cd}.

Now, we find the best partition value of the graph. When the two vertices that define an edge are placed into different sets, or the edge is "cut," we can let this mean that the variable corresponding to that edge is true. So, if you construct this graph for a clause, then the only way to avoid having the endpoints of a large negative edge placed into the same set would be to have one of the edges be cut. So, a boolean variable would have to be true. Therefore, finding the best parition value of such a graph is equivalent to satisfying that 3-SAT clause.

To prevent a variable x_m from being evaluated as true in one clause and false in another, we use the graph in Figure 12 to store the value of the variable each time it is used. An instance of the graph in Figure 12 can be abstracted to an edge of the graph in Figure 11.

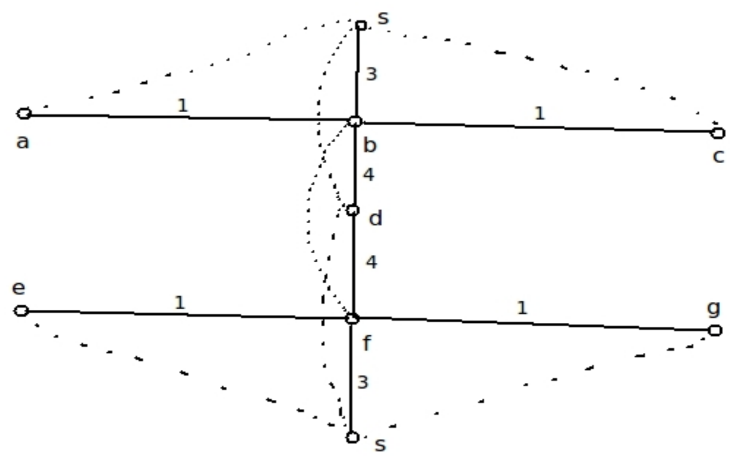


Figure 12: Subgraph for a boolean variable

Again, dotted lines are edges with very large negative weights; for clarity, the vertex s appears twice, once at the top and once at the bottom of the graph. The numbers indicate the weight of the edges.

To denote a boolean variable, we use the pair of edges e_{ab} and e_{bc}, or the pair e_{ef} and e_{fg}. e_{ab} and e_{bc} denote a variable, and e_{ef} and e_{fg} denote its inverse. So, when e_{ab} and e_{bc} are cut, then the variable represented by the graph is true, and when e_{ef} and e_{fg} are cut, then the variable represented by the graph is false. Now, we examine the graph to see if the lowest penalty is obtained when either e_{ab} and e_{bc} are cut or e_{ef} and e_{fg} are cut.

Consider the vertices s, b, d, and f. Because of the strong negative edges e_{sd} and e_{bf}, there are only a few ways to group the vertices so that a negative edge does not incur a cost: e_{sb} and e_{df} are cut, e_{bd} and e_{fs} are cut, or more than two of e_{sb}, e_{df}, e_{bd} and e_{fs} are cut. So, the minimum penalty that can be obtained from these four vertices is 7, which occurs when two edges are cut; when more are cut, the penalty is 10 or higher.

Then, consider the edges e_{ab}, e_{bc}, e_{ef}, and e_{fg}. These edges correspond to the values that the variable evaluates to. For the variable to be consistent, e_{ab} and e_{bc} are cut and e_{ef} and e_{fg} are maintained, or the former are maintained and the latter are cut. In either of these cases, the penalty is 2, plus the minimum penalty from the center four vertices, resulting in a total penalty of 9. If more are cut, then the penalty from these four edges will be greater than 2, resulting in a total penalty greater than 9. If fewer are cut, or one from the first pair and one from the second pair is cut, then the penalty can be as low as zero. However, this would mean that vertices f and b cannot be grouped with s, so the penalty from the center vertices will be 10 or higher. So, the total penalty will be 10 or higher.

Therefore, if the variable is consistent, the penalty is 9. If it is not consistent, it is greater than 9. So, the requirement holds, and this graph can be used as a representation of a variable. We then create multiple copies of the graph, and link them together so that the minimum penalty is obtained when each variable is consistent.

If there are k instances of x_m or $\neg x_m$ in the 3-SAT problem, we create k graphs G_n identical to the graph in Figure 12, with vertices a_n, b_n, and so on. The only difference is that there would be no vertex s_n, keeping s shared for all copies of the graph. Thus, $\forall i, j = a \cdots g, n = 1 \cdots k$ we create vertices i_n and j_n, and set $w(e_{i_n j_n}) = w(e_{ij})$ and $w(e_{i_n s}) = w(e_{is})$. Finally, for all verties b_n and $f_p, \forall n, p = 1 \cdots k$, we set $w(e_{b_n f_p}) = -\infty$. See Figure 13 for an illustration.

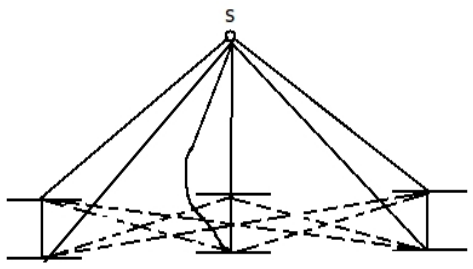


Figure 13: Multiple instances of a variable

We then use the copies of the graph, one for each instance of x_m or $\neg x_m$, linked in the way described.

When we do this, we have a set of graphs that, individually, have the same property as the single graph in Figure 12: the minimum penalty is obtained when one of x_m or $\neg x_m$ is true and the other false. When this minimum penalty is obtained, either vertex b or vertex f is placed in the same group as s.

However, with the repeated, linked graphs, if any b_n is grouped with s, then no f in any other repetition of that graph can be grouped with s, due to the infinite negative edge. Or, if any f is grouped with s, no vertex b can be grouped with s. So, the only way to obtain the minimum penalty is if all vertices b_n or all vertices f_n are grouped with s. Therefore, the only way to obtain the minimum penalty is if, in every copy, e_{ab} and e_{bc} are cut, or if in every copy e_{ef} and e_{fg} are cut. So, the only way to obtain the minimum penalty is if all graphs evaluate the value to true, or all evaluate it to false.

So, we have a system of graphs such that two vertices are placed in the same group if the variable is false and placed in different groups if it is true, and the minimum penalty is obtained when a valid assignment of values is found. Then, by using the graph in Figure 11 to create every clause, we have a graph consisting of k repeated instances of Figure 12. This graph has the property that if the grouped vertices have a penalty equal to $9k$, then we've found that 3-SAT is satisfiable; if it is greater than $9k$, then the instance of 3-SAT cannot be satisfied.

Thus, with an operation that takes polynomial time, we reduced 3-SAT to the vertex grouping problem. So, the vertex grouping problem is NP-Hard.

Each instance of the partitioning problem can be recognized in polynomial time, because the input is just a graph with n vertices and at most $n(n-1)/2$ edges. A solution is a mapping of each element in V to an element in S, so the size of the solution is $O(n)$, one mapping for each element of V. Finally, the measure of a solution can be found in $O(n^2)$ time, by iterating through all of the $O(n^2)$ elements of E and checking if, for any element $e_{ij}, \exists S_k \in S$ such that $i, j \in S_k$. That is, for an edge e_{ij}, we see if i and j are placed in the same set. The sum of the weights of all edges that fail this condition is the penalty incurred by the solution. Because the problem can be recognized in polynomial time, and in polynomial time we can decide if a solution is valid and the measure of a solution, this optimization problem is in NP.

Because the problem is NP-Hard, and is in NP, then the problem of finding the best partition value is NP-Complete.

B Properties of the Observation Evidence Measure

Three elements dominate in determining a numerical measure of the evidence for or against the hypothesis H that two observations refer to the same person. These elements are time, distance, and visual appearance. Specifically, the time lag τ between two observations and the distance δ between the locations where these two observations were made give a lower bound on the speed v at which the observation target must have travelled between these observations:

$$v \geq \frac{\delta}{\tau} \; .$$

A bound that is too high for a given type of target represents evidence against the hypothesis H. For instance, people cannot sustain a velocity of more than 10 m/s for a significant amount of time. Normal walking speed is much lower, about 1.4 m/s. Depending on context, the evidence $\gamma(v)$ for H is high for $v < 1.4$ m/s, drops to zero for a value v_0 somewhere in the 1.4 to 10 m/s range, and diverges to negative infinity as v increases. Positive evidence is never certain, and cannot be allowed to override infinite negative evidence. In summary, including the requirement of monotonicity yields

$$\gamma(0) = 1 \quad , \quad \gamma(v_0) = 0 \quad , \quad v < v' \Rightarrow \gamma(v) \leq \gamma(v') \quad \text{and} \quad \lim_{v \to \infty} \gamma(v) = -\infty \; .$$

The main other factor is similarity of visual appearance, ϕ, a number that is large and positive when similarity is high, and negative when similarity is low. The threshold between positive and negative is determined by two factors: observation noise and object distinctiveness. When the discrepancy d between descriptors of appearance for two observations is of the order of camera noise σ, it provides little or no information, and $\phi(d)$ should be correspondingly close to zero.

Distinctiveness can be defined as a summary statistics β of the differences between observations that are known to correspond to distinct objects. For instance, we can use the mean difference, or some percentile. When the discrepancy d between two appearance descriptors is of the order of this statistics, then the discrepancy is deemed inconclusive, and $\phi(d)$ should be close to zero. Again, we require ϕ to be monotonic, and to go to negative infinity as d increases. In summary,

$$\phi(0) = 1 \quad , \quad \phi(d_0) = 0 \quad , \quad v < v' \Rightarrow \phi(v) \leq \phi(v') \quad \text{and} \quad \lim_{d \to \infty} \phi(d) = -\infty \; .$$

Combination of Measures. To combine noise and distinctiveness considerations, we use

$$d_0 = \max(\beta, \sigma)$$

as the reference value that corresponds to $\phi = 0$.

The value $\gamma(v)$ should be interpreted as *necessary* evidence: when positive, the factor $\phi(d)$ should be left to determine the overall evidence $h(d, v)$ for H, while when $\gamma(v)$ is negative the factor $\phi(d)$ matters less. The reverse should occur as well, in that a very negative $\phi(d)$ should override any finite value of $\gamma(v)$. This is accomplished well and simply by adding the two functions together:

$$h(d, v) = \phi(d) + \gamma(v) \; .$$

C Proof of Condition (2)

Denote the value $w(\mathcal{S})$ of the best partition of a subgraph H as $w^*(H)$, and assume that best-value partitions have been found for subgraphs L and R of G in Figure 1. Then, if we place vertices i and j in the same set of $\mathcal{S}$, the best partition value for G satisfies

$$w^*(G) \leq w^*(R) + w^*(L) + \sum_{e \in C} |\mathrm{w}(e)|$$

because we can construct a partition of the vertices in G by placing the vertices of R and the vertices in L in their optimal partitions, then taking the union of the set of vertices containing i and the set containing j. In the worst case, this partition results in the vertices of every edge with a positive weight being placed in different sets and those of every edge with negative weights being placed in the same set. So, the maximum penalty is the penalty from the partition of R plus the penalty from the partition of L plus the sum of the absolute edge weights in C.

On the other hand, if i and j are placed in different sets of $\mathcal{S}$, then

$$w^*(G) \geq w^*(R) + w^*(L) + \mathrm{w}(e_{ij}) \; .$$

This is because, in the most optimistic case, we partition L and R optimally, and the vertices of the edges in C are partitioned so that the edges in C do not incur a penalty. However, because i and j are placed in different sets, e_{ij} incurs a penalty.

If the minimum penalty from separating i and j is greater than the maximum penalty from keeping them together, then the best partition of G is certain to keep these vertices together. In summary, the vertices i and j at the endpoints of edge e_{ij} are in the same set of $\mathcal{S}$ if the following condition is true:

$$w^*(R) + w^*(L) + \mathrm{w}(e_{ij}) > w^*(R) + w^*(L) + \sum_{e \in C} |\mathrm{w}(e)| \; ,$$

from which inequality (2) follows.

D Complexity of Algorithm 1

We prove that the complexity of Algorithm 1 on page 12 is $O(|V|^2(|V||E|^2 + |V_{max}|^4))$, where V_{max} is the maximum number of vertices in a set of S.

The Algorithm runs the $O(|V'||E'|^2)$ Edmonds-Karp algorithm [4] for every combination of v'_i and v'_j, whose number is proportional to $|V'|^2$. It then runs the Hungarian algorithm to find the bipartite matching between the vertices in S_i and S_j once again for every edge in the matching. Since the number of edges in the matching is at most the size of S_i or S_j, the Hungarian algorithm takes $O(n^3)$ in the number of vertices, which is also the size of S_i or S_j. Thus, the total complexity is $O(|V'|^2(|V'||E'|^2 + |V_{max}|^4))$ in the worst case.

In practice, the complexity is closer to $|V'|^2(|V'||E'|^2) + |V'||V_{max}|^4$, because the flow from one vertex to V' to another is generally very large, except for one or two other vertices. Thus, after calculating the flow, it can usually be seen that the flow is much larger than any edge's value can be, so the next pair of vertices in V' can be considered without running the Hungarian algorithm. Because of this, the number of times that it would be necessary to run it is usually linear in the number of vertices in V'.