Recognition of User-Defined Video Object Models using Weighted Graph Homomorphisms

Dirk Farin[a], Peter H.N. de With[b], Wolfgang Effelsberg[a]
[a]Dept. of Computer Science IV, University of Mannheim, Germany
[b]CMG / Eindhoven University of Technology, The Netherlands

ABSTRACT

In this paper, we propose a new system for video object detection based on user-defined models. Object models are described by "model graphs" in which nodes represent image regions and edges denote spatial proximity. Each node is attributed with color and shape information about the corresponding image region. Model graphs are specified manually based on a sample image of the object. Object recognition starts with automatic color segmentation of the input image. For each region, the same features are extracted as specified in the model graph. Recognition is based on finding a subgraph in the image graph that matches the model graph. Evidently, it is not possible to find an isomorph subgraph, since node and edge attributes will not match exactly. Furthermore, the automatic segmentation step leads to an oversegmented image. For this reason, we employ *inexact* graph matching, where several nodes of the image graph may be mapped onto a single node in the model graph.

We have applied our object recognition algorithm to cartoon sequences. This class of sequences is difficult to handle with current automatic segmentation algorithms because the motion estimation has difficulties arising from large homogeneous regions and because the object appearance is typically highly variable. Experiments show that our algorithm can robustly detect the specified objects and also accurately locates the object boundary.

Keywords: Object recognition, inexact graph matching, image segmentation, dynamic programming.

1. INTRODUCTION AND RELATED WORK

The automatic recognition of user-defined objects has evolved as an important application of image analysis. For example, in object-based video coding applications (e.g. MPEG-4 encoding), user-supplied object models can give hints to the segmentation algorithm about the set of objects to be separated from the background. The user-supplied object model is required to achieve a semantically meaningful segmentation. Other possible applications include queries for video-databases or the derivation of statistics about the occurrence of specific objects over time.

Recently, region-based algorithms for video database access have become popular.[1–3] They are based on finding characteristic regions in a query image. Subsequently, these regions are used to form a database query for images that contain similar regions. However, since the relevant regions are extracted automatically, no pre-knowledge about the spatial object structure is available. Consequently, the object structure is often neglected.

Graph matching is a well-known technique in computer vision and several efficient heuristics have been developed for the graph isomorphism problem. These include algorithms based on nonlinear optimization,[4] quadratic programming,[5, 6] relaxation labeling,[7] or algorithms that are specialized for a specific class of graphs.[8] A completely different approach to region correspondence uses the Earth Movers Distance (EMD), which is a popular distance measure in the field of image retrieval.[9]

2. PRINCIPLE OF REGION-BASED GRAPH MATCHING

Our approach for object detection is based on the assumption that objects can be described reliably by a set of attributed regions and their spatial relationship. The model structure and features are expressed by an object model graph $G_M = (V_M, E_M)$, where each node in V_M represents an image region with uniform color. Nodes have attributes describing region color, shape, and size. Edges in the model graph define spatial proximity. I.e., if $(v_1, v_2) \in E_M$, region v_1 must be near region v_2. The model graph representation allows to recognize objects independent of their exact spatial layout as long as the characteristic spatial structure of the objects remains. In particular, articulated object motion can be modelled in a straightforward way.

Model graphs are defined manually by the user in a graphical editor. To ease the definition, a sample image of the object can be segmented semi-automatically. Subsequently, the features are extracted automatically from the sample image. Finally, the spatial structure of the object is defined by connecting neighboring regions.

Object recognition starts with an automatic color segmentation of the input image. For each of the regions obtained from this segmentation, the same features are extracted as for the model graph. A fully connected *input graph* is defined, taking the regions as nodes again and attributing the edges with the region distances. Since the regions are generated by an automatic segmentation process covering the whole image, this input graph will be much larger than the model graph. Furthermore, due to oversegmentation, regions that belong together semantically may be split into separate regions.

Recognition is based on the idea to find a subgraph in the input graph that matches our model graph. Obviously, it is not possible to find an isomorph subgraph since node and edge attributes will not match exactly and model-graph regions are possibly split into oversegmented regions. Hence, we apply an *inexact* graph matching where several nodes of the image graph can be mapped onto a single node in the model graph ($1 : N$-matching). The quality of a match is described by judging the compatibility of the node and edge attributes.

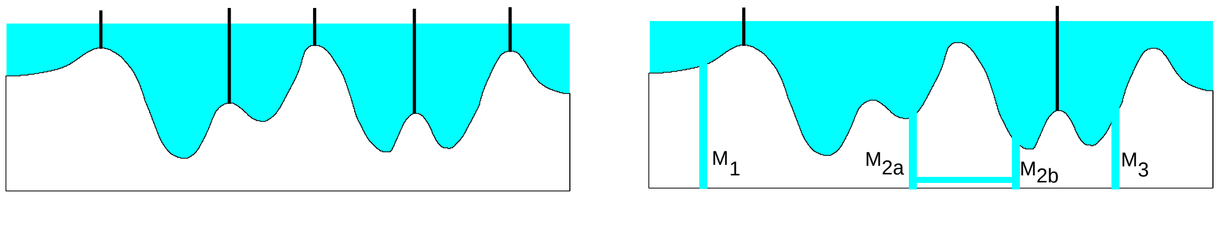
In order to reduce the high computational complexity of graph matching, we employ a fast three-step matching algorithm. The first step reduces the search space by eliminating nodes in the input graph that are very unlikely to occur in the match. The second step performs a $1 : 1$-matching of the *skeleton tree* of the model graph. The skeleton tree is a sub-graph $T = (V_M, E_S)$ (with $E_S \subseteq E_M$) of the model graph that only contains a subset of the edges such that it forms a tree. This $1 : 1$-matching of the skeleton tree can be carried out very efficiently using a dynamic programming approach. The third matching step considers the whole model graph and extends the matching to a $1 : N$-matching.

The paper is organized as follows: Section 3 describes the model editor used to generate the model graphs. The algorithm used for automatic segmentation is outlined in Section 4. Section 5 presents the features used to compare region compatibility and, accordingly, graph similarity. Section 6 delineates the three steps of our matching algorithm. Finally, Section 7 presents results and conclusions are drawn in Section 8.

3. MODEL EDITOR

This section presents an editor for generating model graphs. The object models which are used during the object detection process, are defined manually by the user in a graphical editor. Manual user interaction is required, because only the user knows exactly the semantic meaning of the object model and thus only he can specify the characteristic attributes of a particular object. Since the model specification is an easy task and the models can be saved into a database of frequently used models, the required time for user interaction is low.

Segmentation of the object regions is based on a marker-driven watershed algorithm, which is applied on the gradients of a sample image. The difference to the standard watershed algorithm is that the water does not start flooding from the various local minima. Instead, the water commences to flow from the markers. Several



(a) standard watershed (b) marker driven watershed

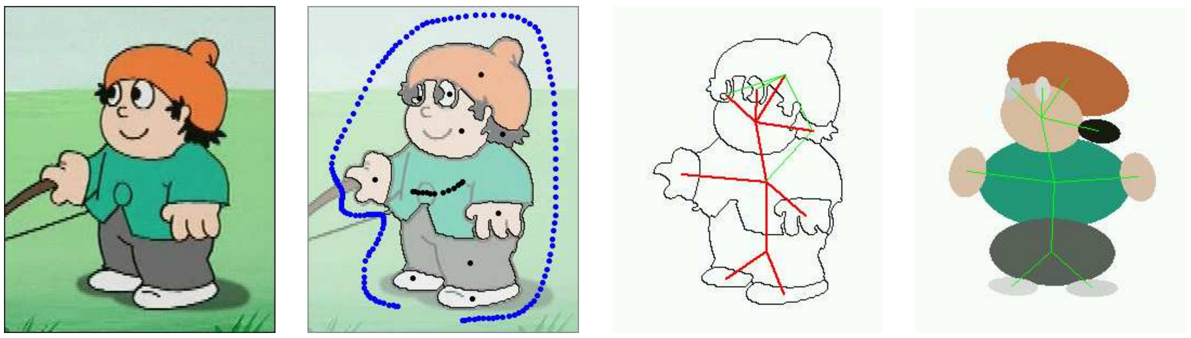
Figure 1. Marker-driven manual watershed algorithm. Whereas in the standard algorithm each local minimum creates a new segment (a), the marker-driven watershed algorithm only builds watersheds between markers (b). Note that the markers M_{2a} and M_{2b} are assigned to the same region, so that no watershed is built between them.

markers can be grouped together such that water basins of these markers are attributed to the same region (i.e., no watershed is built between markers for the same region, see Figure 1).

Relevant object regions are defined manually by placing markers in a sample image. The exact region boundaries are subsequently located by the watershed algorithm. Errors in the segmentation can be corrected by joining regions that have been separated by the watershed algorithm. Internally, this is realized by considering the markers of both regions equivalent (compare the markers M_{2a}, M_{2b} in Figure 1b). The region attributes are extracted from the sample image (cf., Figure 2d), but can be modified by the user in case the sample image does not contain a typical view of the object.

Finally, graph edges are added between the regions that must be close to each other independent of a specific object view. For example, the hand of a human should only be specified as close to the arm, even though it may also be close to the head in a specific sample image.

As we will see later when considering the matching algorithm, matching becomes particularly efficient when the model graph has a tree topology. Therefore, we classify the model graph edges into two classes: *skeleton tree* edges, and *refinement* edges. The principal $1:1$-matching step only uses the skeleton tree edges. This forms no severe restriction, since most natural objects can be described sufficiently using trees. The refinement edges are used in the $1:N$-matching step when oversegmented regions are combined to cover the whole object.



(a) input image (b) manual segmentation (c) model graph (d) *blobbed* model graph showing region features

Figure 2. The creation process of a model graph. Based on the sample input image (a), the user places markers into the image to separate the regions (b). Edges are introduced (c), where model skeleton tree edges are depicted with strong red lines, and the fine green lines denote the refinement edges used in the $1:N$ matching step. The region features can be visualized in an abstract presentation (d).

4. AUTOMATIC SEGMENTATION

Automatic segmentation is carried out using a combination of Watershed segmentation and region merging. The Watershed algorithm provides a very fast pre-segmentation that is strongly oversegmented and thus not sufficient for our purpose. Hence, a region-merging algorithm further combines neighboring regions obtained from the Watershed algorithm. Although the Watershed pre-segmentation is not required, it considerably speeds up the segmentation process, because the region-merging algorithm can start with larger initial regions.

Region merging has proven to be a powerful segmentation algorithm, enabling the use of various merging criteria to control the merging process. We have chosen the Ward criterion, which results in a segmentation in which the region variance is minimized. The fundamental idea is to consider every neighboring pair of regions and calculate the increase of variance that a possible merge of the two regions would impose. Let σ_i^2 denote the variance of region r_i, μ_i the region mean brightness, and $|r_i|$ the region size. Then, the variance increase when regions r_i and r_j are merged, can be calculated by the new variance σ_{ij}^2 minus the original individual variances, which gives

$$\Delta_{ij} = \sigma_{ij}^2 - \sigma_i^2 - \sigma_j^2 = \frac{|r_i| \cdot |r_j|}{|r_i| + |r_j|} \left(\mu_i - \mu_j\right)^2. \tag{1}$$

The region-merging algorithm now successively combines the two regions r_a, r_b for which Δ_{ab} is minimal until the minimum Δ_{ab} exceeds a threshold. We denote the final set of regions as $R = \{r_i\}$. For a complete description of the region merging algorithm used see[10] and.[11]

5. FEATURE EXTRACTION AND MATCHING CRITERIA

To evaluate the similarity of image regions, a set of features is extracted for each region during model creation, as well as for each region generated by the automatic segmentation process. Based on these features, node and edge cost functions are defined which serve as matching criteria in the graph matching step. The calculation of features that are not required for candidate selection (see below) can be postponed after the candidate selection step. Since only a smaller subset of the regions is actually used in the matching process, the computation time is reduced.

5.1. Color

The color of each region is described by its coefficients in the Hue, Value, Saturation (HVS) color space. This color space allows an easy definition of a distance metric having a close relationship with the human perception. HVS space can be visualized as a cone with black at the tip and the rainbow colors around the base. After transforming the HVS color (h, v, s) with $h \in [0; 2\pi]$, $v, s \in [0; 1]$ into cartesian coordinates using $x = v \cdot s \cdot \cos h$, $y = v \cdot s \cdot \sin h$, $z = v$, we use the Euclidean distance between the two colors as color matching cost. We denote the matching cost for assigning region $r_i \in R$ to model node $m_i \in V_M$ as $C_{m_i}^C(r_i)$.

5.2. Size

The shape feature is simply the size of a region in pixels. During the matching process, two cost measures are used for region sizes: one based on the absolute region size and one based on relative region sizes. The absolute region size measure is applied during the candidate selection step to sort out regions that are much larger than the object model. The absolute size feature is computed as the ratio of the input region size $|r_i|$ with respect to the model region size $|m_i|$: $f_{m_i}^S(r_i) = |r_i|/|m_i|$.

Since the size of the object in the image may vary, we do not compare the absolute region sizes to the model in the actual matching step. In fact, only the relative sizes of connected regions are compared to the model. For the computation of the relative size measure, let $|r_i|, |r_j|$ be the sizes of two connected regions and $|m_i|, |m_j|$ the sizes of the corresponding model regions. We define the matching cost $C_{m_i,m_j}^{RelS}(r_i, r_j)$ by the piecewise linear function depicted in Figure 3. This measure does not penalize variations of region sizes up to a factor of two. This is to be robust for varying region sizes because of many factors like occlusions, differing viewing position, deformable objects, or inaccurate segmentation.

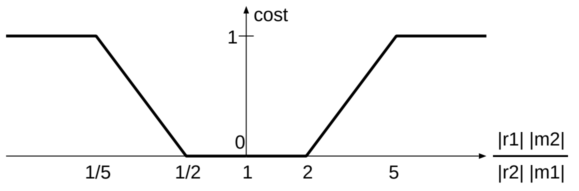


Figure 3. Relative size cost $C_{m_i,m_j}^{RelS}(r_i, r_j)$ for matching a pair of connected model nodes to a pair of input regions.

5.3. Distance

Connected model regions are assumed to have zero distance. However, the distance between a pair of input regions r_i, r_j is measured as the minimum pixel distance $d(r_i, r_j)$ between both region borders. The region distance cost is defined as

$$C^D(r_i, r_j) = \begin{cases} 0 & \text{for } d(r_i, r_j) < d_{min}, \\ \frac{d(r_i,r_j) - d_{min}}{d_{max} - d_{min}} & \text{else.} \end{cases} \tag{2}$$

Truncating the error for small distances has been introduced to tolerate small region distances between input regions. These small distances can be caused by an inaccurate segmentation. We have chosen $d_{min} = 5$ and $d_{max} = 30$ pixels in our experiments, in which the image size was 720×576 pixels.

5.4. Shape

Automatic segmentation usually generates some regions having a "fuzzy" shape, being thin and having many concavities. These regions almost never belong to any object, but rather appear in background regions between objects. Since the regions are often located near object boundaries, they are close to all regions in the object and thus, for the matching algorithm, they seem to be part of the object. Hence, it is preferable to early identify these regions and exclude them from the matching.

To find such "misleading" regions, we make use of a shape feature that describes the *compactness* of a region. It is computed as $f^{Sh}(r_i) = 4\pi \frac{|r_i|}{border(r_i)^2}$, where $border(r_i)$ is the length of the region border. Clearly, the shape feature is maximal ($f^{Sh} = 1$) when the region boundary is a circle and approaches zero when the region is long and thin. Note that f^{Sh} is invariant to scaling.

5.5. Orientation

Edge orientation is an optional matching criterion and can be activated manually for each edge selectively. When matching symmetric objects, the orientation of the matched graph is ambiguous. To break this symmetry, edges can be declared as *oriented* edges. These edges remember which of the two regions is left of the other (or on top of the other). The relative orientation of two regions is determined by comparing the coordinates of their centers of gravity. If the model edge $e = (m_i, m_j)$ is an oriented edge and the orientation of the assigned input regions r_i, r_j differs, additional costs are set to $C^O_{m_i,m_j}(r_i, r_j) = 1$; otherwise we set $C^O = 0$.

5.6. Node and Edge Costs

The above-mentioned costs are combined into node cost and edge cost functions, which are computed by

$$C^N_{m_i}(r_i) = \alpha C^C_{m_i}(r_i), \tag{3}$$

for the node cost and

$$C^E_{m_i,m_j}(r_i, r_j) = \beta C^D(r_i, r_j) + \gamma C^{RelS}_{m_i,m_j}(r_i, r_j) + \theta C^O_{m_i,m_j}(r_i, r_j) \tag{4}$$

for the edge cost, respectively, where the parameters α, β, γ, θ are weighting factors which we have set to 1. They can be increased or decreased depending on the application. When it is a-priori known, for example, that the color may vary because of differing lighting conditions, the weight of the color cost α should be decreased.

For the $1 : N$-matching, we have to generalize the cost measures to handle mappings from several input regions to a single model region. The measures are defined such that the cost measures for $1 : 1$ matching result as a special case. We define the $1 : N$ matching cost for model nodes with $q_i \subset R$ as

$$\hat{C}^N_{m_i}(q_i) = \frac{1}{\sum_{r \in q_i} |r|} \sum_{r \in q_i} |r| \cdot C^C_{m_i}(r). \tag{5}$$

The generalized distance measure computes as (see Fig. 4)

$$\hat{C}^D(q_i, q_j) = \underbrace{\min_{r_a \in q_i, r_b \in q_j} C^D(r_a, r_b)}_{\text{minimum distance between both sets of regions}} + \underbrace{\frac{1}{2} \sum_{r_a \in q_i} \min_{r_b \in q_i, r_b \neq r_a} C^D(r_a, r_b)}_{\text{coherence of region } q_i} + \underbrace{\frac{1}{2} \sum_{r_b \in q_j} \min_{r_a \in q_j, r_a \neq r_b} C^D(r_a, r_b)}_{\text{coherence of region } q_j}. \tag{6}$$

The generalized cost functions for relative region size and orientation $\hat{C}^{RelS}$ and $\hat{C}^O$ are computed by determining the sum of region sizes and the center of gravity for the set of regions. The generalized total edge cost $\hat{C}^E$ is defined as the weighted sum of the individual costs similar to the definition of C^E.

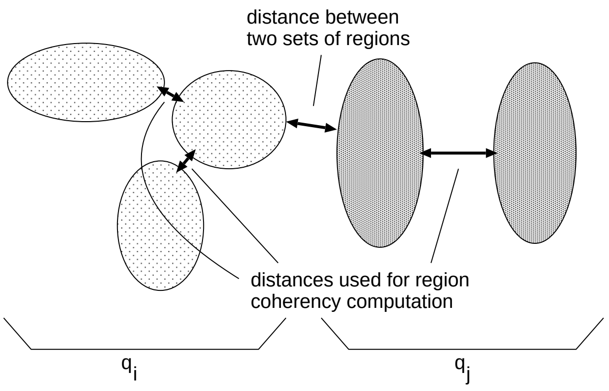


Figure 4. Visualization of computing the distance cost between two sets of regions. The cost is defined as the minimum distance cost of all regions pairs $(r_a \in q_i, r_b \in q_j)$ plus the minimum distance costs between the regions in each set.

6. MATCHING ALGORITHM

Graph matching is carried out in a three-step process. Since the color or size of many of the regions generated by the automatic segmentation will strongly deviate from the model regions, they can be excluded from the matching process to decrease the computational time. The first matching step determines for each model region a subset of input regions and performs the previously mentioned exclusion of unsuitable regions. The second matching step involves only the regions of the selected subset as candidates for a model node. The second matching step computes a $1:1$ matching of the model graph skeleton tree using a dynamic programming approach. This $1:1$ matching acts also as the initialization for the third matching step, where the $1:1$ matching is enriched to form a $1:N$ matching. Enriching means that additional input regions can be assigned to a single model region to decrease overall cost. The three matching steps can also be viewed as incrementally imposing additional structural information. While the first step (candidate selection) is completely free from any structural constraints, the $1:1$-matching obeys the structure of the model skeleton tree, and the final $1:N$-matching step considers the full model graph structure.

6.1. Candidate Selection

The candidate input regions for a model region are selected based on the color, the region size and the shape feature. The idea of the candidate selection step is to sort out regions that have the wrong color, a clearly wrong size, or a non-compact shape. Two selection strategies are possible: we can fix the number of candidates N_C for each model region and select the N_C best input regions as candidates, or we can set a threshold on the region similarity and consider all input regions with higher similarity as candidates. The choice of selection strategy is not critical when the number of candidates are sufficiently high and the thresholds are set high enough to ensure that the correct matches are not sorted out. We adopted a strategy with fixed number of candidates for each model region and observed that about $10-20$ candidates for each model region are sufficient. Formally, we define the candidate selection process as the mapping $c : V_M \times \{1, 2, \dots, |R|\} \rightarrow R$ such that $C_v^C(c(v,i)) < C_v^C(c(v,j)) \rightarrow i < j$ if the region $c(v,i)$ fulfills $C_v^S(c(v,i)) < \theta$ and $f^{Sh}(c(v,i)) < \nu$. Otherwise, the region is placed at the end of the mapping c. Hence, c sorts the input regions according to increasing matching costs. Input regions that are a factor θ larger than the corresponding model region or that have a non-compact shape are sorted out by placing them at the end. In our experiments, we have chosen $\theta = 3$ and $\nu = 0.15$. These values can be adjusted or even selected individually for each model region, depending on the amount this region can change its size in different input images and on the application or model. Since θ and ν are only used to sort out clearly non-matching regions, they can be set arbitrarily large or can even be omitted at all (but more candidates would have to be considered in this case). Since the automatically segmented regions are possibly only part of a single model graph region and the total size of the object to be found in the input image is not known yet, regions that are too small should not be excluded. Note that the same input region can appear as candidate region for several model regions. The constraint that the same input region must not be assigned twice to different model nodes must be obeyed by the following step.

6.2. 1 : 1-Matching

The 1 : 1-matching step is the most important step, since it does the main localization of the model regions. The found matches are the seed for the 1 : N-matching step, where they are further extended with additional input regions.

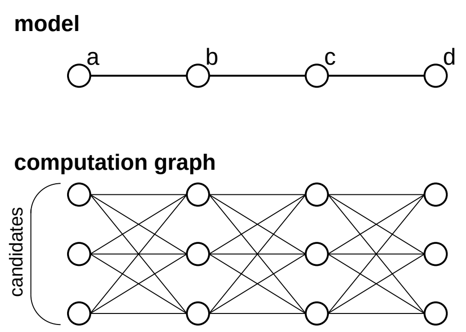
Weighted graph matching can be described as finding the maximum weight clique in the corresponding association graph,[12] which is known to be NP-hard. However, for special classes of graphs, such as e.g. trees, efficient algorithms exist. Since almost all real-world objects can be accurately described by trees and the existence of efficient algorithms for trees, we restrict our 1 : 1-matching step to finding the best matching tree for the skeleton tree of a model graph.

Our algorithm is based on a dynamic programming approach. The objective is to find the mapping $\mathcal{M}_{1:1} : V_M \to \{1, 2, \ldots, |N_C|\}$ that minimizes the sum of node costs and edge costs in the tree:

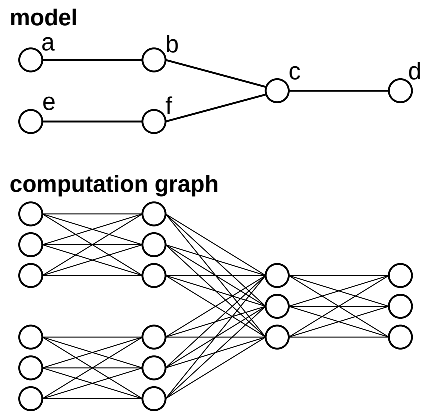
$$\min_{\mathcal{M}_{1:1}} \left\{ \sum_{v \in V_M} C_v^N\big(c(v, \mathcal{M}_{1:1}(v))\big) + \sum_{(v_1,v_2)\in E_S} C^E_{v_1,v_2}\big(c(v_1, \mathcal{M}_{1:1}(v_1))\,,\, c(v_2, \mathcal{M}_{1:1}(v_2))\big) \right\}. \tag{7}$$

Let us introduce the concept of computing the minimum cost mapping with a simple example. Assume that the model tree is e.g. a simple linear chain (Fig. 5a). We construct a computation graph by duplicating each model node to N_C nodes, each representing the decision that the model region is mapped to a specific candidate node. The node costs C^N are assigned to the nodes, i.e., the first column of nodes get costs $C_a^N(c(a,1))$, $C_a^N(c(a,2))$, $\ldots$, $C_a^N(c(a,N_C))$. Similarly, the edge costs C^E are assigned to the edges. Now minimizing the sum (7) is equivalent to computing the minimum cost path through the resulting computation graph. To compute the minimum cost path, we proceed column by column from left to right and calculate for each node the predecessor node that gives the minimum total cost so far. More specifically, we assign attributes *mincost* and *last* to each node in the computation graph. The nodes in the left column are initialized with *mincost* equal to their respective node costs C^N and *last* = **nil**. Continuing with the next column to the right, we calculate for each node the total cost that results from each choice for the predecessor candidate. This cost consists of the *mincost* of the predecessor node, the edge cost C^E linking the predecessor node to the current and the current node cost C^N. The predecessor node that gave the least cost is stored into *last* and the corresponding minimum cost in *mincost*. When we arrive at the rightmost column, the candidate with the minimum cost is selected and the minimum cost path is traced back using the *last* attributes.

If the model tree contains junctions like shown in Figure 5b, the algorithm above has to be extended. Since model node c has multiple incoming edges, the best predecessor candidate has to be selected from both, model



(a) model graph is a chain



(b) model graph with a single join

Figure 5. Object model skeleton trees with their respective computation graphs.

Algorithm 1 Initialize the computation graph for subsequent dynamic programming algorithm.

```
Require: the model tree $V_M = \{v_1, \ldots, v_N\}, E_S \subset V_M \times V_M$
 1: $pred(v_1 \in V_M) \leftarrow \emptyset$
 2: $l \leftarrow \{v_1\}$
 3: $r \leftarrow V_M \setminus \{v_1\}$
 4: while $l \neq \emptyset$ do
 5:    select an arbitrary $v \in l$ and set $l \leftarrow l \setminus \{v\}$
 6:    for all $(v, w) \in E_S \wedge w \in r$ do
 7:       $pred(v) \leftarrow pred(v) \cup \{w\}$
 8:       $l \leftarrow l \cup \{w\}$
 9:    end for
10: end while
```

Algorithm 2 Computing the minimum cost assignment.

```
 1: procedure $calccolumn(v \in V_M)$
 2: for all $w \in pred(v)$ do
 3:    call $calccolumn(w)$
 4: end for
 5: for $n = 1$ to $N_C$ do
 6:    if $pred(v) = \emptyset$ then
 7:       $mincost(v, n) \leftarrow C_v^N(v, c(v, n))$
 8:    else
 9:       $cost \leftarrow C_v^N(v, c(v, n))$
10:       for $w \in pred(v)$ do
11:          $cost \leftarrow cost + \min_i\big(mincost(w, i) + C_{w,v}^E(c(w, i), c(v, n))\big)$
12:          $last(v, n, w)$ is set to the $i$ that minimized the above sum
13:       end for
14:       $mincost(v, n) \leftarrow cost$
15:    end if
16: end for
```

node b and model node f. Consequently, *mincost* is now obtained by minimizing over the sum of all previous nodes and incoming edge costs. The computation time required therefore increases from N_C^2 steps for a column to $indegree \cdot N_C^2$ computations ($indegree = 2$ in our example). However, the total computation time does not increase, because the total number of edges in the computation tree remains constant. Hence, the complexity is $O(N_C^2 \cdot |V_M|)$. The complete matching algorithm is described in Algorithm 1 and 2. Algorithm 1 initializes the *pred* attributes that define the order in which the model nodes have to be considered in the calculation. Algorithm 2 performs the actual $1:1$ matching. It is a recursive algorithm which is started with $calccolumn(v_1)$.

In each junction node, we now have to store not only a single predecessor, but the best candidate node for each incoming model tree edge. When tracing back the minimum cost path, this is no longer a linear chain but rather a minimum cost tree.

The algorithm described thus far has still one drawback. When the same input region occurs as candidate for different model nodes, the algorithm may use the same input region more than once. This is not desirable. For example, consider searching for a human with equal left and right arm (see Figure 6). The model nodes for both arms are the same and both sub-trees are connected to the same body node. Since either the left or right arm in the input graph will match better to the model, the algorithm will assign the best one to both arms of the model.

This problem can be alleviated using two techniques. First, it is possible to make the edges connecting the arm and the body *oriented* edges (see Section 5.5) inducing extra cost when the left arm in the input graph is mapped to the right arm in the model graph and vice versa. However, this does not work in all situations and we have to extend the algorithm described above to prevent double assignments. This can be done by introducing another attribute *blocked* to each computation graph node. This attribute stores the set of input regions that are used so far. In each junction node v of the computation graph ($|pred(v)| > 1$), combinations of previous node candidates k_1, k_2 that collide ($blocked(k_1) \cap blocked(k_2) \neq \emptyset$) cannot be selected. Note that since this is a combinatorial problem, the best candidate node for all preceding nodes cannot be determined independently. In fact, all combinations are enumerated and checked for validity. The valid combination with the minimum cost defines the best candidates for the preceding model nodes.

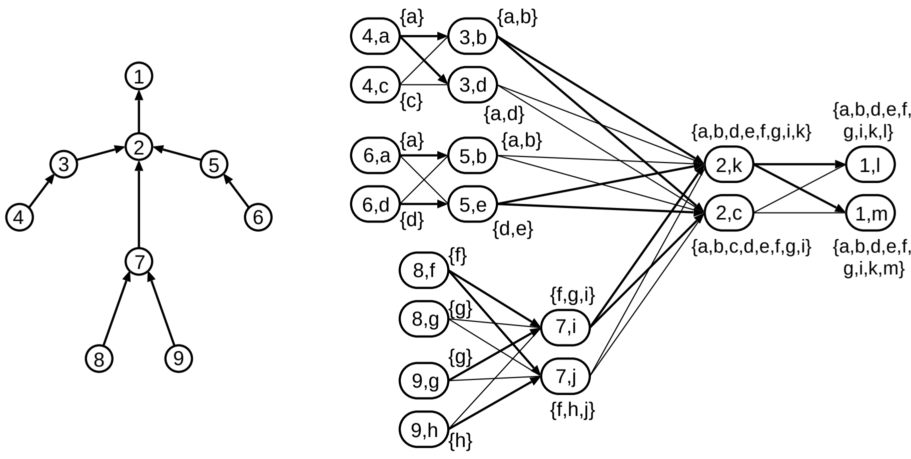


Figure 6. Example calculation for a model graph (left) describing a human. The arrows denote the order of calculation as induced by the *pred* attributes. For simplicity, the computation graph on the right has been constructed with only two candidate nodes for each model node. The model nodes are denoted by numbers and the input nodes by letters. Selected edges are drawn with thick arrows and the corresponding *blocked* attribute is shown at each node. Calculation proceeds from left to right.

As an example, consider Figure 6. Note that node 3 selects input regions a and b as its left arm. Under the assumption that the right arm looks identical to the left arm in the model, the dynamic programming algorithm without *blocking* attribute would select the same input regions for the right arm. But since a and b are contained in the *blocked* set of node 3b and node 5b, node 2 has to choose nodes d and e for the right arm. However, it cannot ensure that both arms are assigned to the correct side, because the orientation is lost in the graph description. This orientation ambiguity can be resolved by making the edges connecting the arms with the body *oriented* edges.

6.3. $1 : N$-Matching

In the third step, additional input regions are assigned to model nodes if this decreases the total cost. We define the $1 : M$ matching through a mapping $\mathcal{M}_{1:N} : V_M \to 2^R$. It is initialized with the result of the preceding $1 : 1$ matching by

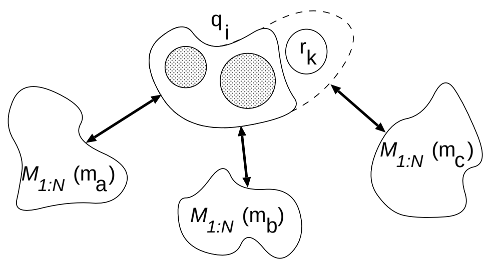
$$\mathcal{M}_{1:N}(m_i) := \{c(m_i, \mathcal{M}_{1:1}(m_i))\}. \tag{8}$$

More input regions are added with a greedy algorithm. In each iteration, a cost difference $\delta_{m_i}(q_i, r_k)$ is computed which equals the cost difference that would be induced when input region r_k is added to region set q_i. As long as the cost difference is below zero, the input region with the largest decrease of cost is added. Otherwise, the algorithm ends. We define the cost difference as

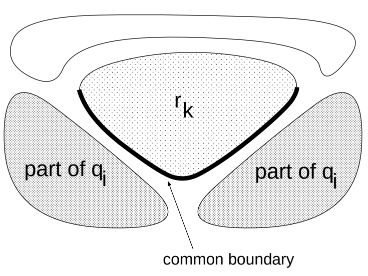
$$\begin{aligned}\delta_{m_i}(q_i, r_k) = &\underbrace{\hat{C}^N_{m_i}(q_i) + \sum_{(m_i,m_j)\in E_M} \hat{C}^E_{m_i,m_j}(q_i, \mathcal{M}_{1:N}(m_j))}_{\text{old node and edge costs for region set } q_i} \\ &\underbrace{-\hat{C}^N_{m_i}(q_i \cup \{r_k\}) - \sum_{(m_i,m_j)\in E_M} \hat{C}^E_{m_i,m_j}(q_i \cup \{r_k\}, \mathcal{M}_{1:N}(m_j))}_{\text{new node and edges costs for regions set } q_i \text{ plus region } r_k} - \underbrace{\epsilon \hat{C}^B(q_i, r_k)}_{\substack{\text{cost reduction because} \\ \text{of common boundary}}}\end{aligned} \tag{9}$$

where the first terms are the generalized node and edge costs as defined in Section 5.6. The last term $\hat{C}^B$ decreases the cost for region r_k if it shares a common boundary with the regions in q_i (cf., Fig. 7). The weighting factor ϵ was set to 0.5 in our experiments.

To calculate $\hat{C}^B(q_i, r_k)$, we consider each pixel on the boundary of r_k and search for the region r' that is nearest to the pixel among all candidate regions for all model nodes. $\hat{C}^B(q_i, r_k)$ is set to the fraction of pixels for which the nearest region $r' \in q_i$. Clearly, if r_k is completely surrounded by regions in q_i, then $\hat{C}^B(q_i, r_k) = 1$.



(a) testing region r_k for inclusion into region set q_i



(b) common boundary between r_k and q_i

Figure 7. (a) In $1:N$ matching, the hypothetical cost after adding input region r_k is computed. If the cost difference is lower, r_k is attributed to the model node. (b) Definition of the common boundary of a region, i.e., the part of a boundary that is *inside* the region set.

Finally, we can summarize the effects of the $1:N$ matching step with a simple rule: a region will be added to the set of assigned regions of a model node if

- the region is mostly surrounded by other regions mapped to the same model node,
- the region bridges the space between two regions that should be neighboring, or
- the combined region size (or color) matches better to the model node size.

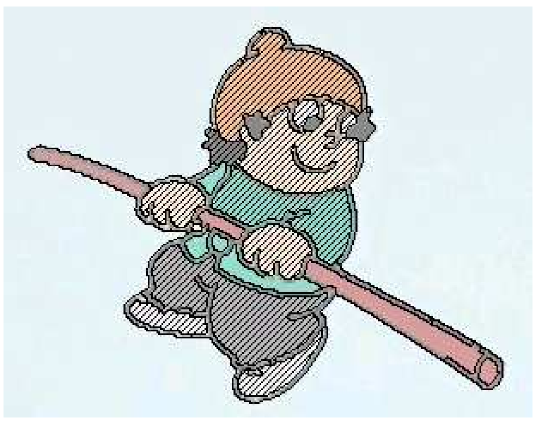
Figure 8 portrays that $1:N$ matching improves the accuracy of the object model detection. Since model regions have been split into several parts by the occlusion of a foreground object, $1:N$ matching is required to cover the whole model region.

7. EXPERIMENTS AND RESULTS

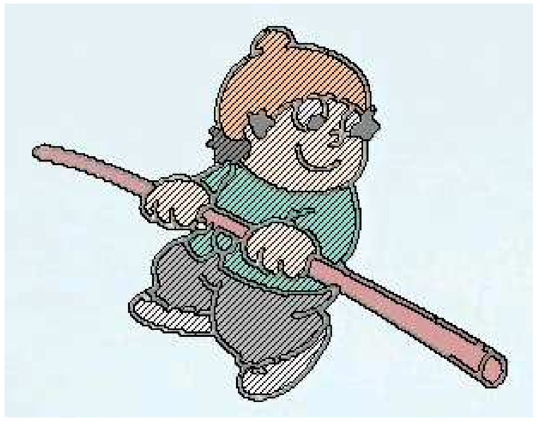
An example matching result for a scene with several objects having similar characteristics is shown in Figure 9. The object defined by the model is detected reliably. Small errors occur at the left hand of the object since the algorithm cannot decide whether the fingers are part of the hand or not. As the size of the hand without fingers matches better to the size of the jacket, the fingers are discarded.

Further experiments revealed that the matching algorithm is very robust. Possible errors are mostly introduced due to a erroneous color segmentation. Sometimes, regions with equal color are combined into the same region when they are close to another. Since the matching algorithm can map several input regions to a single model region, but not several model regions to a single input region, the matching algorithm searches for another region instead of assigning the undersegmented input region twice.

The computation time is currently about 1 second for a 720×576 video frame on a 550 MHz Pentium-III processor. Most of the time is used for the color segmentation step. Hence, it is possible to test the same input image for several object models even faster as the initial color segmentation only has to be computed once.



(a) 1 : 1-matching



(b) 1 : N-matching

Figure 8. Matching the object model from Fig. 2. The image shown is part of a larger input image with several other objects. Fig. 8a shows the result after matching the object skeleton tree. Matched regions are marked. Note that the jacket is not covered completely since it has been oversegmented into several regions. Fig. 8b depicts the matching results after the 1 : N-matching step. After the 1 : N matching, the jacket is completely covered.

8. CONCLUSIONS

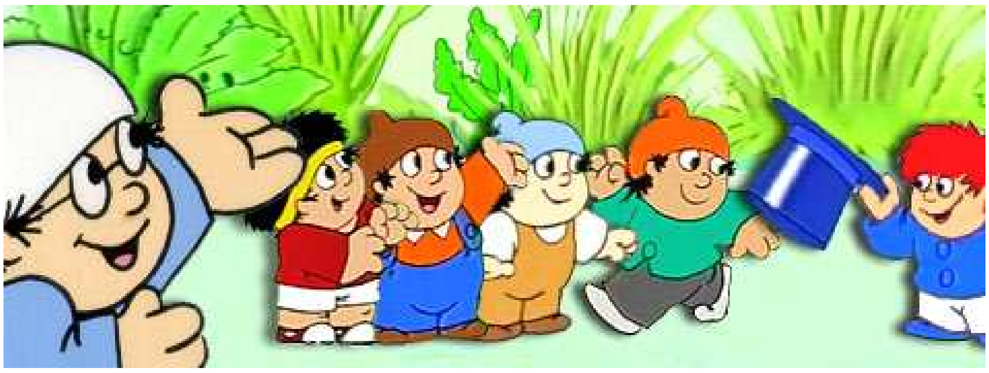
We have presented a new algorithm for the detection of video objects that are described by manually defined object models. This enables to find the object even when it is deformed by articulated motion. The central matching step is carried out using a dynamic programming algorithm, which is fast and accurate. An advantage of our model-based object detection algorithm is that it does not rely on a preceding object segmentation algorithm. Both object segmentation and object recognition are performed in a combined framework. This eliminates the need for accurate object segmentation masks, which are required e.g. in shape-based object recognition algorithms.[13]

Up to now, we have processed mainly cartoon images[*] since this allows to use a simple color-based algorithm for segmentation. Further research will be conducted to generalize the region definition and features to support textured regions. This will make the algorithm applicable to natural images. The performance of region definition is still limited at the moment. However, the current implementation can be used to identify well-defined objects, such as logos in natural video sequences.

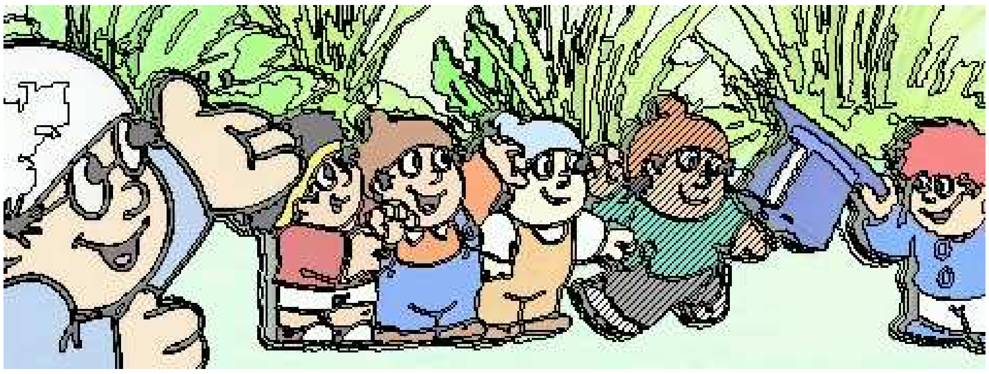
REFERENCES

1. C. Carson, S. Belongie, H. Greenspan, and J. Malik, "Region-based image querying," in *CVPR'97 Workshop on Content-Based Access of Image and Video Libraries*, 1997.
2. J. Li, J. Z. Wang, and G. Wiederhold, "IRM: Integrated region matching for image retrieval," in *ACM Multimedia*, 2000.
3. Y. Chen and J. Z. Wang, "A region-based fuzzy feature matching approach to content-based image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , pp. 1252–1267, 2002.
4. S. Gold and A. Rangarajan, "A graduated assignment algorithm for graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , 1996.
5. C. Schellewald and C. Schnörr, "Subgraph matching with semidefinite programming," tech. rep., CVPR, University of Mannheim, Nov. 2002.
6. M. Pelillo, K. Siddiqi, and S. W. Zucker, "Continuous-based heuristics for graph and tree isomorphisms, with application to computer vision," in *NIPS 99 Workshop on Complexity and Neural Computation*, Dec. 1999.

[*]The video images shown in this paper have copyright by ZDF (Zweites Deutsches Fernsehen), Germany.



(a) input image



(b) found object model (second from the right)

Figure 9. The object model shown in Figure 2 is searched for in an input image with several similar objects.

7. A. Torsello and E. R. Hancock, "Efficiently computing weighted tree edit distance using relaxation labeling," in *Energy Minimization Methods in Computer Vision and Pattern Recognition, Third International Workshop, EMMCVPR 2001, France*, *Lecture Notes in Computer Science* **2134**, Springer, Sept. 2001.
8. D. Eppstein, "Subgraph isomorphism in planar graphs and related problems," tech. rep., Dept. of Information and Computer Science, University of California, May 1994.
9. H. Greenspan, G. Dvir, and Y. Rubner, "Region correspondence for image matching via EMD flow," in *IEEE Workshop on Content-based Access of Image and Video Libraries*, June 2000.
10. D. Farin and P. H. N. de With, "Towards real-time MPEG-4 segmentation: a fast implementation of region merging," in 21st *Symposium on Information Theory in the Benelux*, pp. 173–180, 2000.
11. T. Brox, D. Farin, and P. H. N. de With, "Multi-stage region merging for image segmentation," in 22nd *Symposium on Information Theory in the Benelux*, May 2001.
12. M. Pelillo, K. Siddiqi, and S. W. Zucker, "Matching hierarchical structures using association graphs," tech. rep., Yale University, Center for Computational Vision & Control, Nov. 1997.
13. S. Richter, G. Kühne, and O. Schuster, "Contour-based classification of video objects," in *SPIE Proc. Storage and Retrieval for Media Databases, 4315*, pp. 608–618, Jan. 2001.
14. R. Fablet, P. Bouthemy, and M. Gelgon, "Moving object detection in color image sequences using region-level graph labeling," in 6th *IEEE International Conference on Image Processing, ICIP*, Oct. 1999.