Contribution to T1 Standards Project

STANDARDS PROJECT:	Coding and Performance Specifications for Multimedia Communications on Internet Services (T1A1-15).
TITLE:	Medium Bandwidth Techniques for Estimating Temporal Delays Between Input and Output Video Sequences
SOURCE:	NTIA/ITS
CONTACT:	Margaret Pinson Voice: (303) 497-3579 Fax: (303) 497-5323 e-mail: margaret@its.bldrdoc.gov Stephen Wolf Voice: (303) 497-3771 Fax: (303) 497-5323 e-mail: steve@its.bldrdoc.gov
DATE:	May, 1999
DISTRIBUTION:	Working Group T1A1.5
KEY WORDS:	video delay, temporal alignment, medium bandwidth features
ABSTRACT:	This contribution presents new techniques for estimating temporal delays between input and output video streams from video teleconferencing systems. The discussion in this contribution considers "variable alignment," defined as an alignment process that can assign a unique alignment offset, or delay, to every output video frame in a given output video sequence. To achieve the bandwidth reduction necessary for in-service measurements, the variable alignment techniques presented in this contribution are applied to subsampled video images. We have found that a sub-sampling factor of 128 to 1 produces useful variable alignment results. Regardless of the sub-sampling factor used (including sub-sampling factors of 1), there is always a possibility of misalignment. The probability of misalignment increases as the amount of scene motion decreases and the video distortion present in the output increases. Therefore, it has been necessary to implement artificial intelligence techniques that examine the set of most probable alignments produced by the signal processing portion of the algorithm. All of the techniques presented in this contribution can also be applied to whole field correlation algorithms such as those given in ANSI T1.801.04.

Medium Bandwidth Techniques for Estimating Temporal Delays Between Input and Output Video Sequences

1. Introduction

ANSI T1.801.04-1997 [1] describes a "variable alignment" technique where each output video field can have a unique alignment offset, or transmission delay. The variable video alignment technique described in ANSI T1.801.04 requires comparisons of output video fields to input video fields. Hence, this technique is computationally expensive and difficult to implement as an in-service[1] measurement since perfect copies of the sampled output video fields must be sent to the transmission system input (or vice versa).

On the other hand, ANSI T1.801.03-1996 [2] and a companion contribution [3], describe "constant alignment" techniques where each output video field in the output sequence is assigned the same fixed alignment offset, or transmission delay. These constant alignment techniques estimate video delay by correlating low bandwidth feature streams that are extracted from the input and output video (e.g., temporal information, or TI). Since these extracted features are very low bandwidth, they may easily be communicated in real time between the input and output ends using commonly available ancillary data channels. Hence, constant alignment techniques are suitable for real-time in-service measurements. Constant alignment techniques track the average video delay of the output video sequence being examined and are thus insensitive to delay variations of individual video fields.

The purpose of this contribution is to present a real-time in-service video delay algorithm with the desirable attributes of both variable alignment (each output video field can have a unique video delay) and constant alignment (the ancillary data channel bandwidth required to perform the delay measurement is much less than the original video stream). To achieve the bandwidth reduction necessary for in-service measurements, the variable alignment techniques presented in this contribution are applied to subsampled video images. We have found that a sub-sampling factor of 128 to 1 (sub-sampling NTSC *fields* by 16 in the horizontal direction and by 8 in the vertical direction) produces useful variable alignment results.

Regardless of the sub-sampling factor used (including sub-sampling factors of 1), there is always a possibility of misalignment. The probability of misalignment increases as the amount of scene motion decreases and the video distortion present in the output increases. Therefore, it has been necessary to implement artificial intelligence techniques that examine the set of most probable alignments produced by the signal processing portion of the algorithm. All of the techniques presented in this contribution can also be applied to whole field correlation algorithms such as those given in ANSI T1.801.04.

[1] The term "in-service" is used in the sense that the input and output ends of the video transmission/storage system under test are not at the same physical location, and there is no *a priori* knowledge of the input video.

2. Overview of Variable Alignment Algorithm

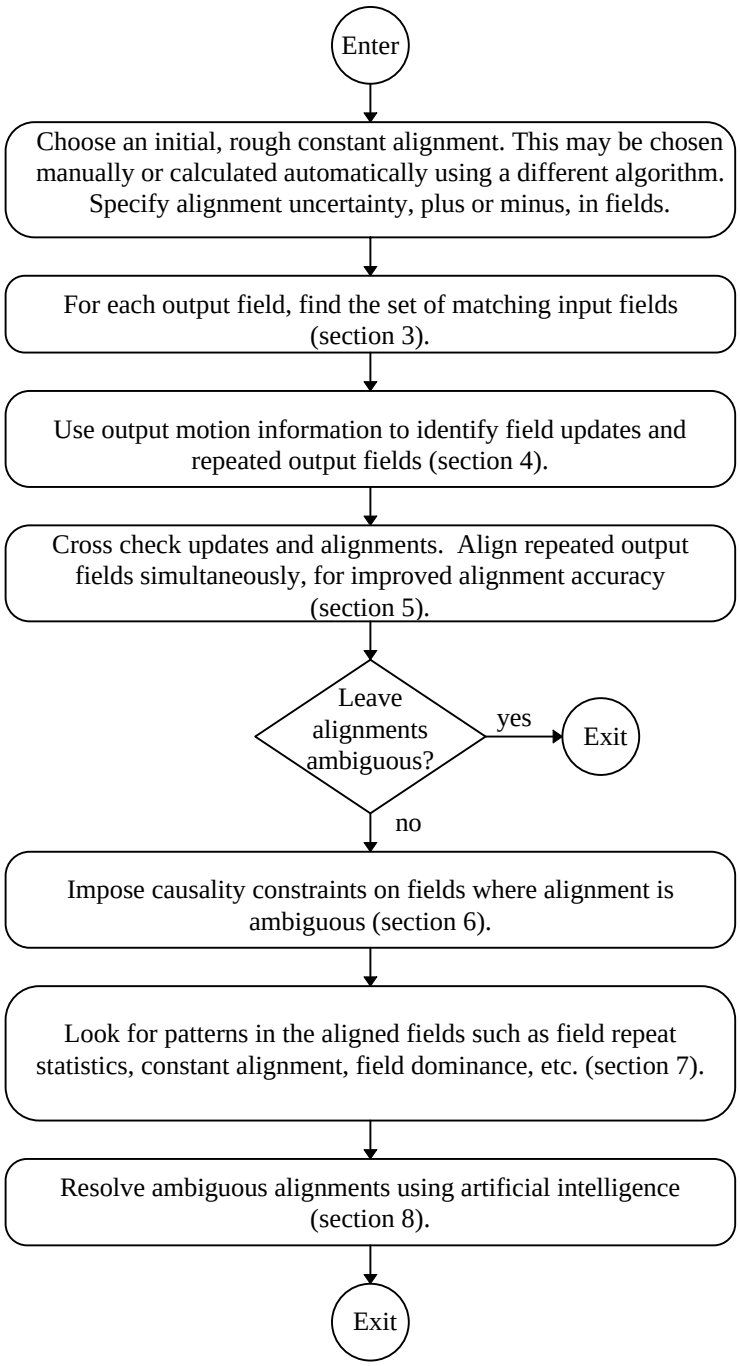

Figure 1. Variable alignment overview.

Figure 1 presents an overview of the variable alignment algorithm. The algorithm requires a initial alignment guess and an alignment uncertainty to limit the amount of searching that is performed. The initial alignment guess can be obtained from a constant alignment algorithm [2, 3] and the uncertainty can be estimated as one half of the maximum amount of delay variation.

For each output field, the likely set of matching input fields is determined by correlating with subsampled input images. The shape of the correlation function determines how many input fields are included in the likely set of matching input fields. A flat correlation function (such as produced when the motion is very limited) results in more matching input fields than a sharp correlation function.

Output motion information is used to identify probable output field updates and repeated fields. Using field update information to cross check the correlation alignment information enables one to discard those input fields from the set of matching input fields that do not produce consistent alignments.

At this point, some alignments might still be ambiguous. The algorithm can be terminated or a higher level artificial intelligence (AI) algorithm can be used to deduce the most likely alignments. This AI algorithm imposes causality constraints and looks for patterns in the aligned fields. For example, examination of the data might reveal that one NTSC field type (e.g., field 1) is much more likely than the other and hence alignments to the other field type might be discarded. All output fields

aligned by AI techniques are tagged by the action that was performed so that these can be distinguished from alignments determined in earlier phases of the algorithm.

3. Correlation Alignment of Each Output Field

This section describes the process that is used to select a set of matching input fields for each output field. These matching input fields are retained for further consideration in later stages of the algorithm. To reduce the ancillary data channel bandwidth required to implement the variable alignment algorithm, the output to input field correlations are performed using subsampled luminance images.

3.1 Subsampled, Normalized Luminance Field Generation

Before aligning fields, the valid viewing region of each field must be subsampled and normalized. The output video must also have been corrected for spatial shifts. However, the output level shift and gain do not have to be corrected since the correlation function that is being used here is insensitive to these quantities. In this description, the luminance field is noted as Y, and the time n where this field occurs is denoted as t_n. Pixels of Y are further subscripted by row and column, i and j respectively, so that an individual pixel is denoted as $Y(i,j,t_n)$.

Pixels outside of the active video area are discarded. Assuming a 720 pixel by 243 line NTSC field, pixels outside of the center 672 pixels and 224 lines are also discarded; this approximates the region displayed on a television monitor. Then, further lines and pixels are discarded until the number of lines divides evenly by SUBSAMP_LINES and the number of pixels divides evenly by SUBSAMP_PIXELS.

An un-normalized subsampled image (UNS_n) is formed by dividing the remaining pixels of Y into rectangular regions, each containing SUBSAMP_LINES lines and SUBSAMP_PIXELS pixels, and taking the mean of each rectangle. For most scenes, choosing SUBSAMP_LINES=8 and SUBSAMP_PIXELS=16, for a bandwidth compression factor of 128, produces good variable alignment results.

To normalize the subsampled image at time *n*, divide the subsampled image by the standard deviation (stdev) of that image's pixels, namely

$$subsamp_n = UNS_n / \text{stdev}(UNS_n).$$

3.2 Definitions

best is the alignment offset of the closest matching input field.

best_one is the alignment offset of the closest matching input field of NTSC field type 1.

best_two is the alignment offset of the closest matching input field of NTSC field type 2.

consider is the current set of matching input field alignments.

CS_i is the value of the correlation function for input field *i*.

CS_THRES_DIFF is the correlation threshold when comparing different NTSC field types.

CS_THRES_SAME is the correlation threshold when comparing the same NTSC field types.

$in_subsamp_i$ denotes the normalized, subsampled input field number *i*.

$out_subsamp_j$ denotes the normalized, subsampled output field number *j*.

3.3 Correlation Alignment Algorithm Flowchart

Figure 2 shows the processes used to determine the set of matching input fields for each output field. At the end of this flowchart, the input fields that are most similar to this output field have been identified. If exactly one input field remains, the alignment of the output field is certain, and the field is *aligned*. If two or more input fields remain under consideration at the end of this flowchart, then the alignment of the output field is *ambiguous*.

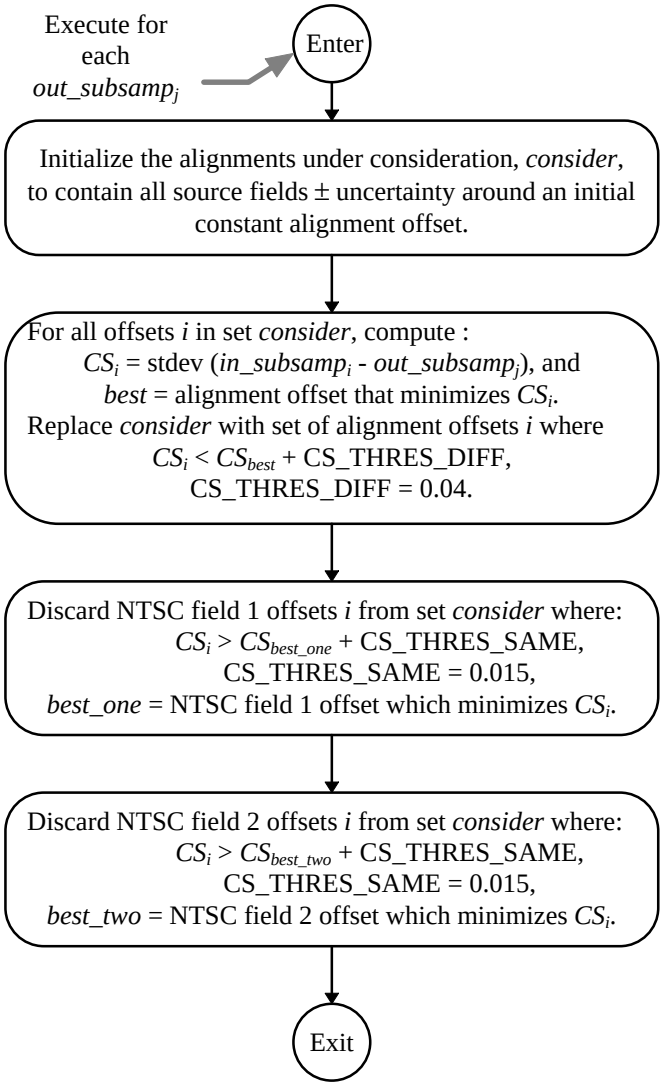

Figure 2. Algorithm to align one output field.

Aligning each field separately requires three correlation steps. First, the output field is correlated with all source fields under consideration. Second, the output field is correlated to only NTSC field type 1 input fields. Third the output field is correlated to only NTSC field type 2 input fields. All of these correlation steps use the standard deviation of the difference between the normalized input and the normalized output, for which smaller values represent better alignments.

For all three steps, correlations function values close to the minimum correlation value are considered to be equally valid. The closeness magnitude tests are implemented using an additive confidence threshold (CS_THRES_DIFF, CS_THRES_SAME), with recommended values 0.04 and 0.015 as shown in the flowchart. When comparisons include different fields (NTSC field type 1 vs. NTSC field type 2) the confidence threshold must be greater because of possible spatial-temporal differences between these two fields. Namely, a video system which field repeats a field 1 into a field 2 may perform a spatial-temporal interpolation to the latter field to account for the different sampling of these two fields. As a result, the correlation values CS_i are often slightly higher when comparing different NTSC field types than when comparing the same NTSC field types. Generation of interpolated output fields by the video system is one contributor of alignment uncertainty since these output fields do not have a matching input field.

3.4 Alignment Correlation Function

All of the correlation steps for determining output field alignment use the standard deviation of the difference between the normalized input and the normalized output subsampled images.

Fundamentally, this is the same correlation function used to time align feature streams for constant alignment [2, 3], except that here we are applying it to fields. Using this correlation function for output to input field alignment has several advantages. The correlation function is easy to compute and is not sensitive to DC level or gain variations in the output video. Intuitively, the correlation function finds the point of maximum cancellation of the signal variance. The image normalization process described in section 3.1 is helpful because it compensates for reduced contrast in the output video when blurring is present.

3.5 Outcome of the Algorithm

At the completion of the flowchart in Figure 2, the one or more input fields which most closely resemble each output field have been identified. Ambiguous alignments are common when the output video is extremely distorted or when the input video is nearly still.

Each field is represented by a subsampled luminance image. If the recommended subsampling values are used, these subsampled images will require approximately 1 Mbits/sec of ancillary data channel bandwidth to make in-service video delay measurements. Subsampling factors may be increased or decreased according to the ancillary data channel bandwidth that is available. Measurement accuracy as a function of ancillary data channel bandwidth will be fully examined at a later time.

4. Locate Field Updates

This step identifies sequential output fields which align to a common input field or to a common input frame. This behavior may be loosely termed “field repeating” and “frame repeating” respectively. While section 5 will distinguish between output fields which align to a common input field versus output fields which align to a common input frame, this section does not differentiate between these two cases.

Contiguous sequences of identically aligned output fields will be used to reduce alignment uncertainty. The amount of motion present in the output video can be used to fairly reliably differentiate field updates (i.e., new field) from field repeats (i.e., repeat of previous field). However, the identification of field updates and field repeats is approximate since no one algorithm has yet been found to work perfectly for any amount of scene motion, video system output distortions, scene noise level, and measurement instrument sampling noise. The field classifications produced by this step will be made more robust in Section 5, which incorporates additional information to improve their reliability.

4.1 Output Field Motion Detection

The active video area must be determined before computing *motion*, a Boolean array that specifies which output fields are updates. The subsequent calculations described here are only performed within this active video area. First, compute the temporal information for each pixel using output field $Y(t_n)$ and the previous output field of the same NTSC field type $Y(t_{n-2})$ as

$$TI2(i,j,t_n) = | Y(i,j,t_n) - Y(i,j,t_{n-2}) |.$$

Next, compute

$$MOVING(i,j,t_n) = \{ 1 \text{ if } TI2(i,j,t_n) > MOTION_THRESHHOLD, \text{ and } 0 \text{ otherwise } \},$$

where

$$MOTION_THRESHHOLD = 30.$$

Then compute

$$FRACTION(t_n) = mean_{space} [MOVING(i,j,t_n)],$$

which is the fraction of pixels at time t_n where TI2 detects a moderate to large amount of motion. FRACTION(t_n) is then converted to the Boolean *motion(t_n)* as follows:

motion(t_n) = { **true** if (FRACTION(t_n) >= MOTION_FRACTION), and **false** otherwise },

where

$$MOTION_FRACTION = 0.00006$$

which is true when at least 0.006% of the valid region contains moderate to large amounts of motion. When *motion(t_n)* is **false**, that field is very likely to be a repeated field (i.e., identical to the previous field of the same NTSC field type). Conversely, when *motion(t_n)* is **true**, that field is likely a be a true field *update*. The setting of ***

With the above algorithm, a 15 frames per second codec would have the following pattern: "… false, false, true, true, false, false, true, true, …" Notice that the field update is detected twice, once by NTSC field type 1 and once by NTSC field type 2. The difference between a given field and the immediately preceding field (e.g. NTSC field type 1 and NTSC field type 2) is not used. This is because the different spatially-temporal sampling locations of these two fields can introduce errors into the field update measurement.

4.1.1 Motion Detection Thresholds

In our experience, the MOTION_THRESHHOLD should be set between 20 and 50 to produce good field update detection performance. As the threshold is reduced, the probability of missing an update is reduced but the probability of falsely detecting a field repeat as an update is increased. The default value of 30 represents a reasonable compromise between these two types of errors for the scenes that we have been using. The optimal setting of MOTION_THRESHHOLD is clearly scene dependent. Lower thresholds can be used to detect smaller amounts of motion provided the scene has a higher signal to noise ratio. One must consider all sources of noise in the output scene when setting the MOTION_THRESHOLD (e.g., compression noise, tape noise, NTSC encoding noise, and sampling noise). The reader is referred to [4] for additional information and techniques that may be helpful for determining the proper motion detection threshold that will differentiate true pixel motion from output scene noise.

The MOTION_FRACTION serves to make the measurement more reliable. At the default motion threshold of 30 for a region size of 672 pixels by 243 lines, the default value for MOTION_FRACTION requires that at least ten pixels be above the motion threshold for the field to be declared an update. This prevents a single abnormal pixel difference from controlling the measurement but enables small amounts of motion to trigger the update.

4.2 Definitions

motion$_j$ is a Boolean that specifies if output field *j* is an updated field; see Section 4.1.

updates$_j$ contains the position and length of the field updates. For each output field *j*:

If *updates$_j$* = 0, then field *j* is identical to field *j*-2. These two fields align to the same input field.

If *updates$_j$* = 1, then field *j* appears to be different from both field *j*-2 and *j*+2. Field *j* has a unique alignment.

If $updates_j > 1$, then fields j, j+1, j+2, …, $(j+updates_j -1)$ all align to the same input field, but field j-2 aligns to a different field. This field is an update followed by one or more repeats. The value of $updates_j$ is the number of sequential output fields which align to the same input field.

update_rate is an estimation of the system's field update rate based solely upon the array *motion*. The value of *update_rate* is a whole number, representing the number of output fields which align to one input field. This number is chosen on the high side and is equal to 1 for systems that do not exhibit any field repeating.

4.3 Field Updates Algorithm Flowchart

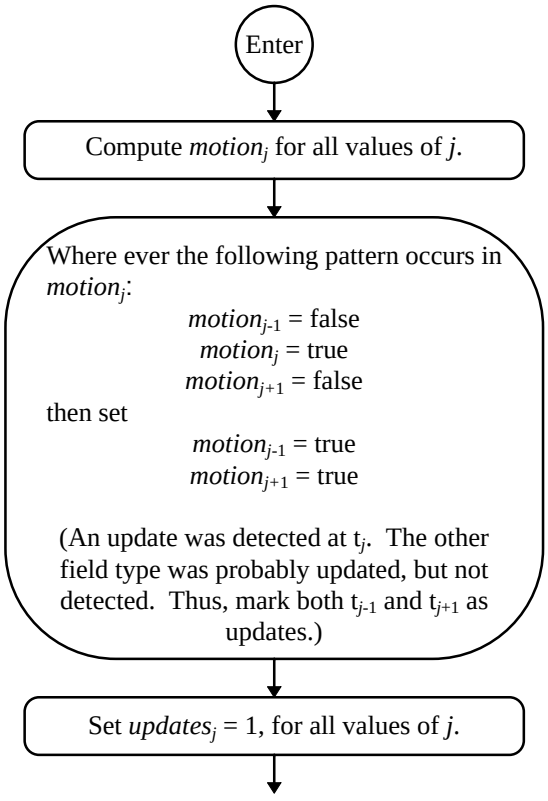

Figure 3. Locate field updates.

Figure 3 and Figure 4 show the process used to identify the location of field updates and runs of repeated fields. The variable *motion* identifies fields which are probably updates. Whenever an update is detected in one NTSC field type but not the other NTSC field type (e.g. the update is detected in NTSC field type 1, but not in the fields of NTSC field type 2 immediately preceding and following), then an ambiguous case has occurred. While this update pattern is unlikely, it may occur due to either a false update detection on one field type or a missed update on the other field type. In our test data, both of these cases occurred. To be safe, the algorithm assumes that both the field before and the field after the detected update are also updated fields.

With that special case handled, the next step is to locate updated fields followed by one or more output fields with identical input alignments. Variable *updates* holds this information and it is established by applying pattern matching to variable *motion*.

At this point recall that the purpose of detecting field updates is not to precisely determine the field repeat rate of the video system. Rather, the goal is to use field update information to improve the reliability of alignment. With that goal in mind, a likely or probable rate of updates is computed. Updates with lengths more than twice the probable update rate may occur, but often this is indicative of a missed update. Updates in near still portions of video sometimes fall below the recommended motion thresholds and are thus missed.

If the probable update rate (i.e., *update_rate* in Figure 4) is "every field has a unique alignment" (i.e., *update_rate* = 1), then the algorithm assumes that all repeated fields are errors. Otherwise, the *updates* that exceed twice the probable update rate are split into two or more smaller updates, each at least as long as the probable update rate.

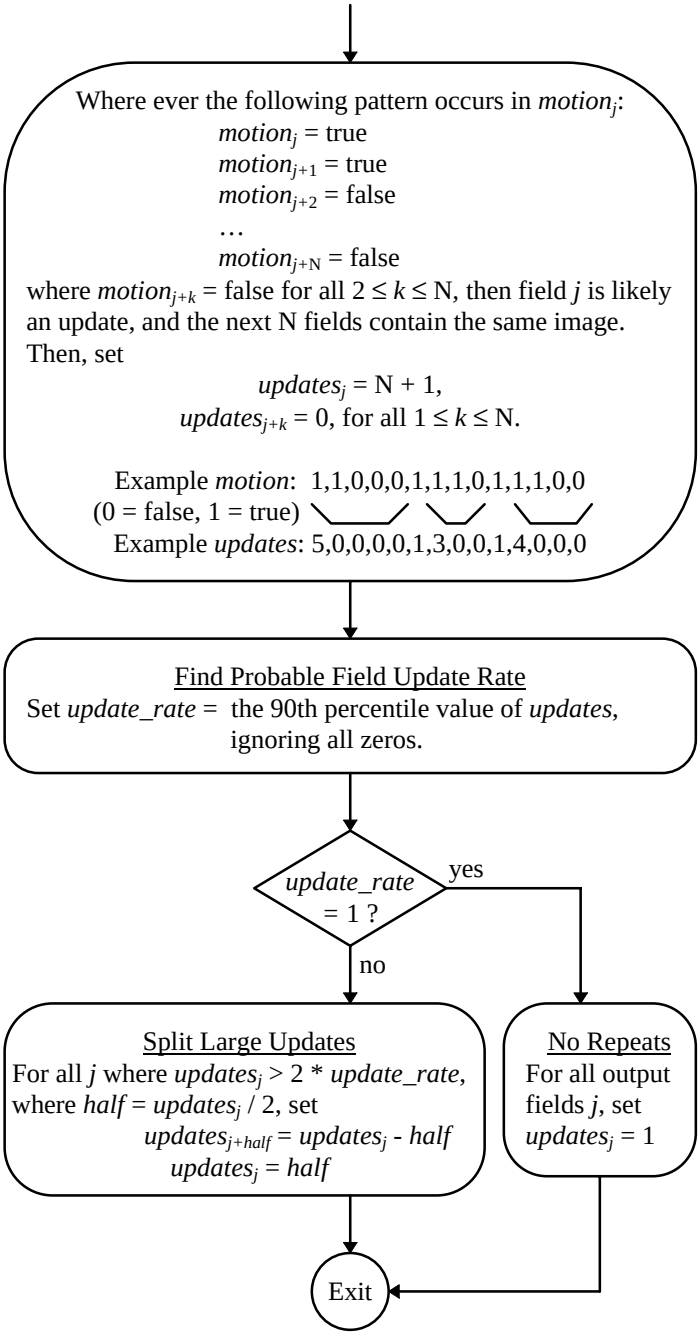

Figure 4. Locate field updates (continued).

If an undetected update occurred, then splitting the long update will be a more accurate representation of the data. If an update really did not occur, little harm is done. For example, a codec that usually issues updates at a six field interval may issue fourteen identical fields immediately preceding a scene cut. The length fourteen update would be split into two length seven updates. The knowledge that each of these seven fields probably align to the same input field can still be used to greatly improve the alignment calculations of those seven fields in later stages of the algorithm.

4.4 Outcome of Algorithm

At the completion of the algorithm shown in Figure 3 and Figure 4, a pattern of updated fields and repeated fields has been identified. Because there were no comparisons of unlike field types (NTSC field type 1 vs NTSC field type 2), there is no way to determine whether the fields that are grouped by this algorithm align to an input field or an input frame. This ambiguity will be addressed later in the contribution.

The final pattern of updates will include errors, both false detections and missed updates. One example of a falsely detected update is a field update that is detected by three consecutive fields, instead of two consecutive fields. For example, the codec outputs a single field six times, but the first one is ever so slightly different than the rest. In this instance, the first output field of the sequence of six fields would be mistakenly labeled as having a unique alignment and thus *updates* = { 1, 5, 0, 0, 0, 0}, when the correct update pattern is really = {6, 0, 0, 0, 0, 0}. This may occur in noisy areas of the scene when the motion detection thresholds are too low and noise is detected as motion.

Missed updates will falsely suggest that a series of output fields all align to a single input field when in fact one of the supposedly repeated output fields is an update, and the output fields after

this update align to a different input field. For example, *updates* = { 10, 0, 0, 0, 0, 0, 0 , 0, 0, 0} when the pattern should be {6, 0, 0, 0, 0, 0, 4, 0, 0, 0}. This may occur in very low motion areas of the scene when the motion detection thresholds are too high to detect movements that are nonetheless perceptible to a viewer. The last two steps of the flowchart in Figure 4 seek to reduce the probability and impact of such errors.

5. Cross-check Updates with Correlation Alignments

This section combines the field alignments from the correlation algorithm in section 3 with the field update pattern from section 4. Both sets of data are improved through bi-directional cross checks. Identical output fields are averaged and correlated for improved alignment accuracy. One alignment is selected for each sequence of two or more identical output fields.

5.1 Definitions

$updates_j$ contains the position and length of field or frame updates, from section 4.

$field_updates_j$ will be filled with the position and length of **field** updates. Otherwise, this variable is the same as $updates_j$. Set *field_updates* = *updates*. For each output field *j*:

- If $updates_j = 0$, then field *j* is identical to field *j*-1. These two fields align to the same input field.
- If $updates_j = 1$, then field *j* is different enough from both *j*-1 and *j*+1, that a common alignment cannot be assumed (i.e., field *j* is a unique output field).
- If $updates_j > 1$, then fields *j*, *j*+1, *j*+2, …, (*j*+$field_updates_j$ -1) all align to the same input field, but field *j*-1 aligns to a different field. The value of $field_updates_j$ is the number of sequential output fields which align to the same input field.

$choices_j$ is the number of input fields that could align to output field *j*, from section 3.

$where_{j,i}$ is the possible input field alignments for output field *j*, numbered according to timecode and sorted as follows: $where_{j,i} < where_{j,i+1}$, $1 \le i \le choices_j$, from section 3.

intersect is the set of input fields where each input field in the set is a candidate alignment for all of the output fields in sequence being examined.

CS_THRES_AVE is the correlation threshold when comparing averaged output NTSC fields.

5.2 Cross-check Algorithm Flowchart

Figure 5 shows the algorithm used to perform the cross-checks between the correlation alignment data from section 3 and the field update patterns from section 4. Notice that, for simplicity, the definition of the update pattern is narrowed here to include only cases where the sequence of output fields appears to align to one input field. This may be loosely termed "field repeating".

First, the field update pattern is checked against the correlation alignment data. This check identifies output field sequences that were falsely said to identically align to one input field, when in fact these output fields align to two or more input fields. When an error is detected, each output field in the sequence is re-classified as a unique output field. This check sometimes (but not always) detects instances where one or more field updates have been missed by the updates location algorithm in section 4. When small but perceptible differences in image content are insufficient to trigger the MOTION_FRACTION threshold in section 4, the field alignments from the correlation algorithm may indicate that an update has occurred. Also, this check will

sometimes (but not always) detect sequences of output fields that align to one input frame instead of one input field.

Next, correlation alignment data are checked against the new field update pattern. For each sequence of output fields that (according to the new field update pattern) align to one input field, this check eliminates those input fields that are not a potential alignment for all of the output fields. The later steps in this flowchart (i.e., Figure 6) are only appropriate for sequences of output fields that can all align to one input field.

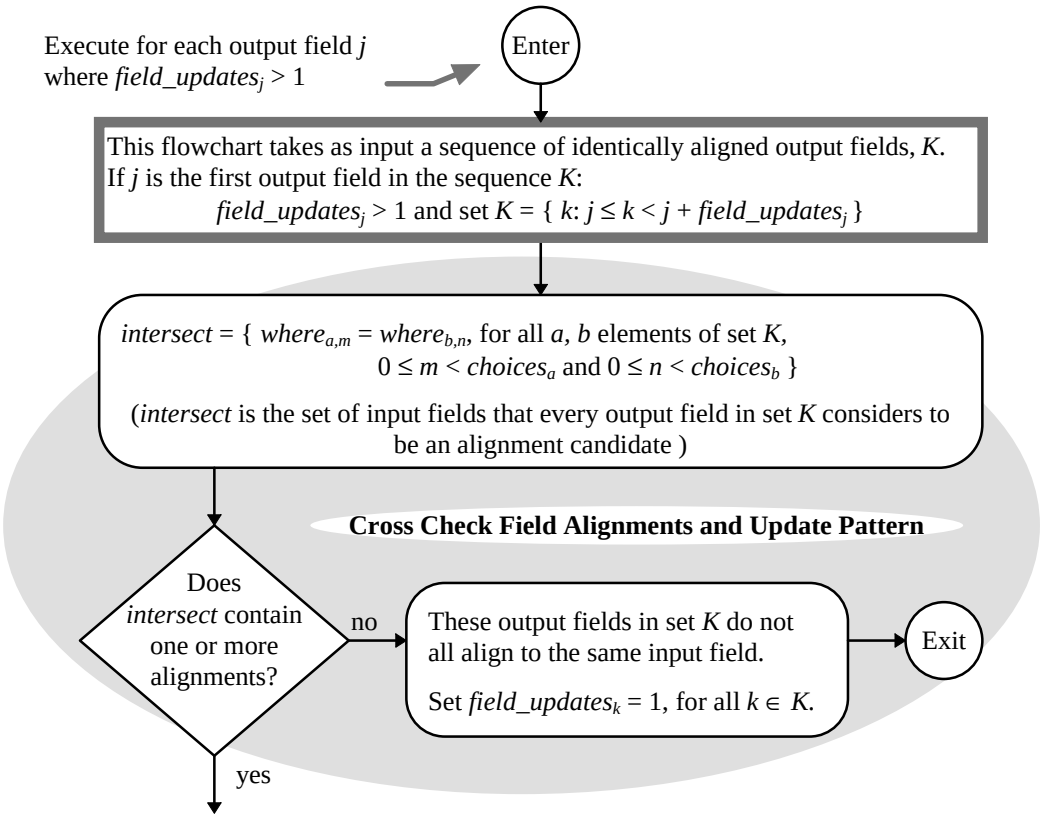

Figure 5. Cross-check alignment of identical fields.

In Figure 6, output fields are averaged for greater alignment accuracy and one alignment is chosen for each sequence of output fields that all align to the same input field. The algorithm takes advantage of the knowledge that several output fields align to a single input field. The normalized, subsampled output images that contain NTSC field type 1 are averaged together to form an averaged field type 1 image. This averaged field type 1 image is then correlated with the input fields under consideration. The process is repeated for NTSC field type 2 output images. Note that the spatial-temporal differences between the two NTSC field types prevent us from averaging them both together. This averaging process is thus applied to output fields of each NTSC field type separately.

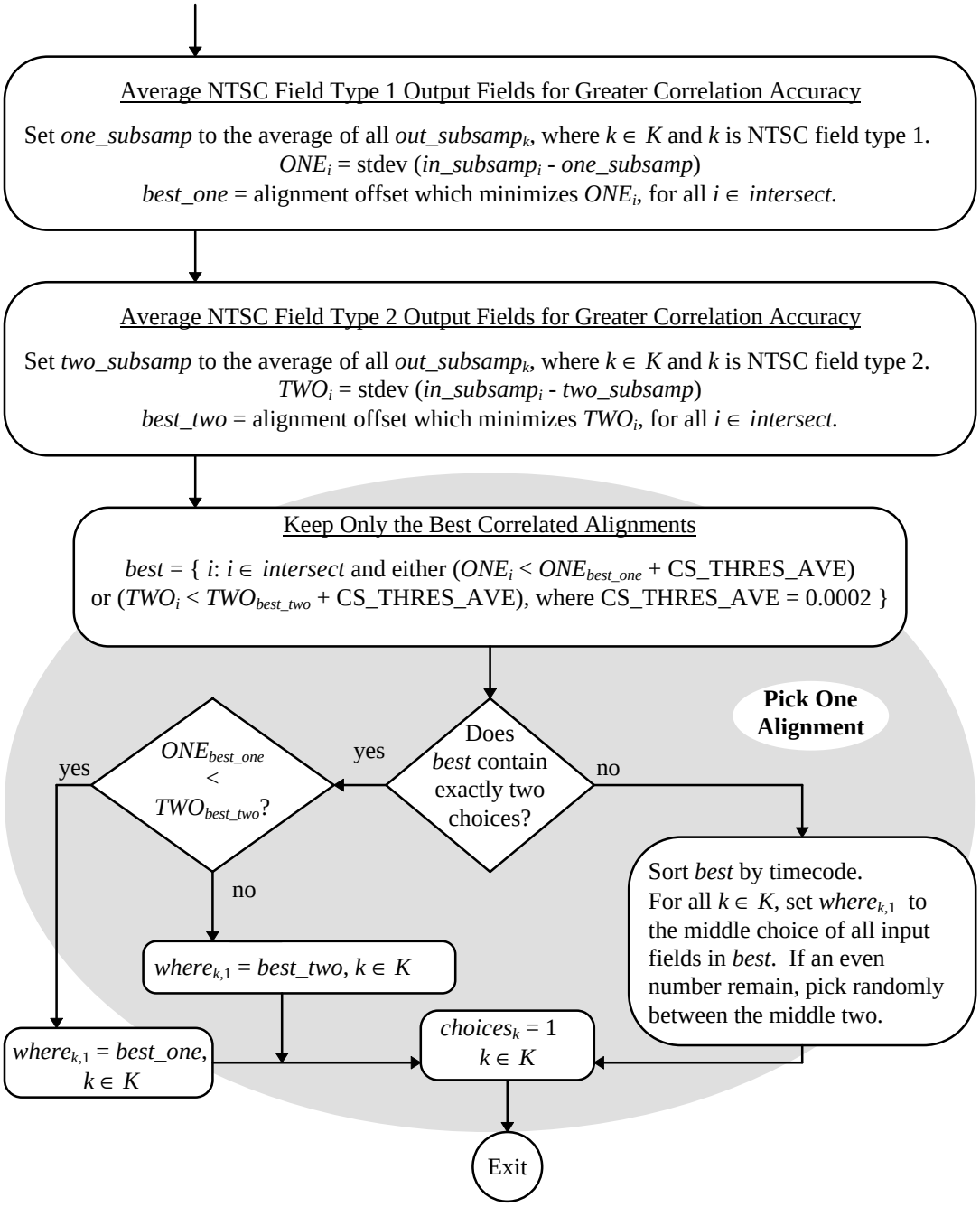

Figure 6. Cross-check alignment of identical fields (continued).

An extremely tight correlation uncertainty threshold (CS_THRES_AVE) with a recommended value of 0.0002 is applied to each correlation curve as shown in Figure 6. The averaging of output fields reduces noise and improves the accuracy of the correlation curve, which is one reason for the extremely tight uncertainty at this point in the algorithm. Another reason is that we desire to choose one alignment for the set of output fields by the end of this flowchart. The extremely tight correlation criteria forces the algorithm to select only those input fields that are very close to the minimum correlation value for either NTSC field type 1 or NTSC field type 2.

After the preceding step, one or more valid input alignments remain. If exactly two alignments remain, then the best alignment choice is made deterministically based on the lowest point in either of the two correlation curves. If an odd number of input alignments remain, then the middle alignment is chosen. For example, if the input alignments remaining were 02:00:58:12, 02:00:58:12*, and 02:00:58:13 (here * indicates NTSC field type 2 and the absence of a * indicates NTSC field type 1, where field 1 occurs earlier in time than field 2), then input field 02:00:58:12* would be chosen as the best alignment. If only one input alignment remains, it is thus chosen. If an even number of input fields remain (four or greater), then the best alignment is randomly selected between the middle two alignments.

5.3 Outcome of Algorithm

At the completion of the algorithm in Figure 5 and Figure 6 for a sequence of output fields that all align to the same input field, either that sequence has been re-classified so that each field has a unique alignment, or one input field has been selected to align to all the output fields in that sequence. Unique output fields may still have an ambiguous alignment. These output fields may or may not have an alignment different from both the preceding and following field. These intermittent ambiguous alignments may be resolved by choosing the middle alignment as outlined above, or they may be left ambiguous (as shown in Figure 1). Alternatively, artificial intelligence may be used to further align these ambiguous output fields. This last option will be considered in greater detail next.

6. Eliminate Non-Causal Alignments

Taken together, sections 6, 7 and 8 describe one method for resolving remaining alignment ambiguities (i.e., output fields j where $choices_j$ is not equal to one). This section describes an optional step that imposes causality, a reasonable assumption for most video systems. The algorithm finds and eliminates non-causal alignments from further consideration. Causality constraints are imposed on each NTSC field type separately.

6.1 Definitions

$choices_j$ is the number of input fields that could align to output field j, from section 5.

$where_{j,i}$ is the possible input field alignments for output field j, numbered according to timecode and sorted as follows: $where_{j,i} < where_{j,i+1}$, $1 \le i \le choices_j$, from section 5.

$ambiguous_field_j$ is an output field j for which $choices_j \neq 1$.

6.2 Causality Algorithm Flowchart

Figure 7 shows the process that is used to impose causality constraints on the remaining ambiguous alignments. First, forward causality is imposed such that the earliest possible input alignment of an output field must be at or past the earliest possible input alignment of the previous output field of the same NTSC field type. Next, backward causality is imposed such that the latest possible input alignment of an output field must be at or before the latest possible input alignment of the next output field of the same NTSC field type. Note that in both cases, we might be left with no choices for some output fields.

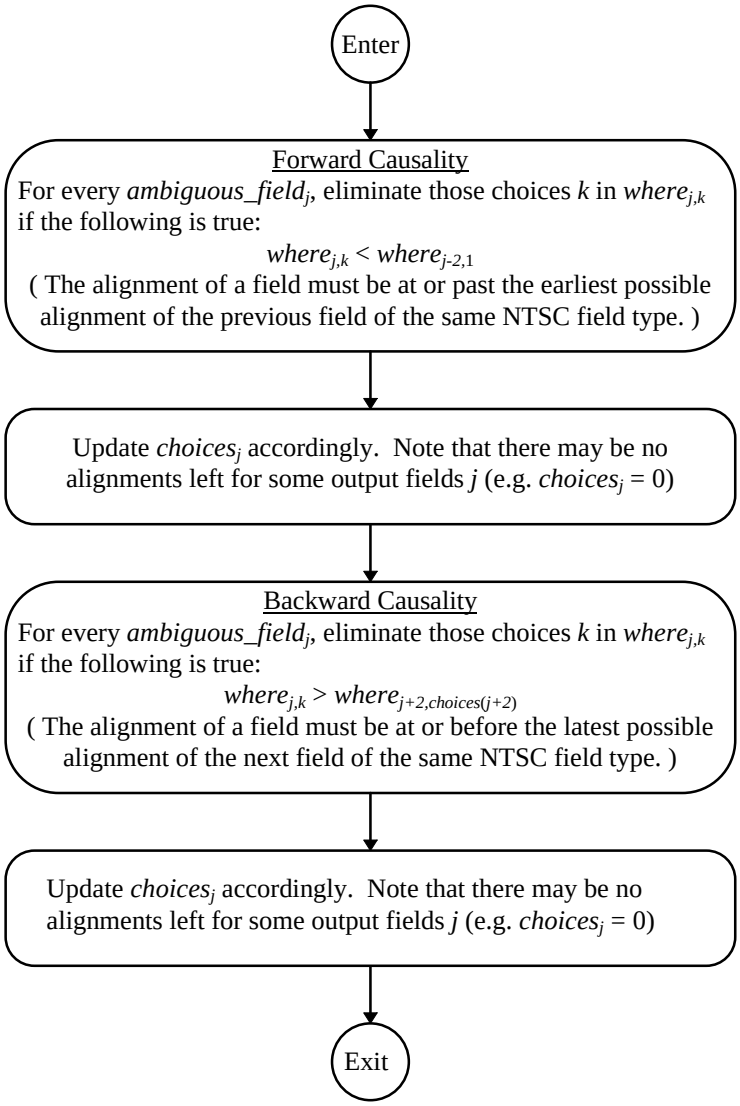

Figure 7. Impose Causality Constraints.

7. Gather Statistics on Unambiguously Aligned Fields

This algorithm in this section determines field repeating patterns and statistics for the unambiguously aligned output fields (i.e., those output fields j that have $choices_j = 1$). These patterns and statistics will then be used to deduce likely alignments for the ambiguous cases.

7.1 Definitions

total is the total number of output fields in the scene.

$choices_j$ is the number of input fields that could align to output field j, from section 6.

$where_{j,i}$ is the possible input field alignments for output field *j*, numbered according to timecode and sorted as follows: $where_{j,i} < where_{j,i+1}$, $1 \le i \le choices_j$, from section 6.

$align_j$ contains the input field to which output field *j* aligns if $choices_j = 1$; undefined otherwise.

$aligned_field_j$ denotes an output field *j* for which $choices_j = 1$.

$ambiguous_field_j$ denotes an output field *j* for which $choices_j \neq 1$, from section 6.

known_align is the total number of unambiguously aligned output fields in the scene.

$rough_const_j$ is the alignment of output field *j* assuming an initial constant alignment that is an input to this algorithm (see Figure 1).

Statistics Gathered in Algorithm

f1_more_likely is true if 90% or more of $aligned_field_j$ align to input fields of NTSC field type 1, provided at least 2% of *total* output fields are aligned (weak *f1* dependence).

f2_more_likely is true if 90% or more of $aligned_field_j$ align to input fields of NTSC field type 2, provided at least 2% of *total* output fields are aligned (weak *f2* dependence).

f1_dominant is true if 95% or more of $aligned_field_j$ align to input fields of NTSC field type 1, provided at least 20% of *total* output fields are aligned (strong *f1* dependence).

f2_dominant is true if 95% or more of $aligned_field_j$ align to input fields of NTSC field type 2, provided at least 20% of *total* output fields are aligned (strong *f2* dependence).

choose_constant is true if a constant alignment [3] is chosen.

Variables Used to Define Each Contiguous Sequence of Ambiguous Fields

before and *after* are the input fields that are aligned to the output fields immediately before and after the ambiguous section, respectively.

start and *stop* are output field numbers noting the beginning and ending of the ambiguous section.

before_repeat is the number of contiguous output fields prior to *start*, all of which align to input field *before*.

7.2 Statistics Gathering Algorithm Flowchart

Figure 8 and Figure 9 shows the processes used to gather patterns and statistics on aligned output fields. These statistics and patterns are then used by the algorithms in section 8 to choose likely alignments for the ambiguous fields.

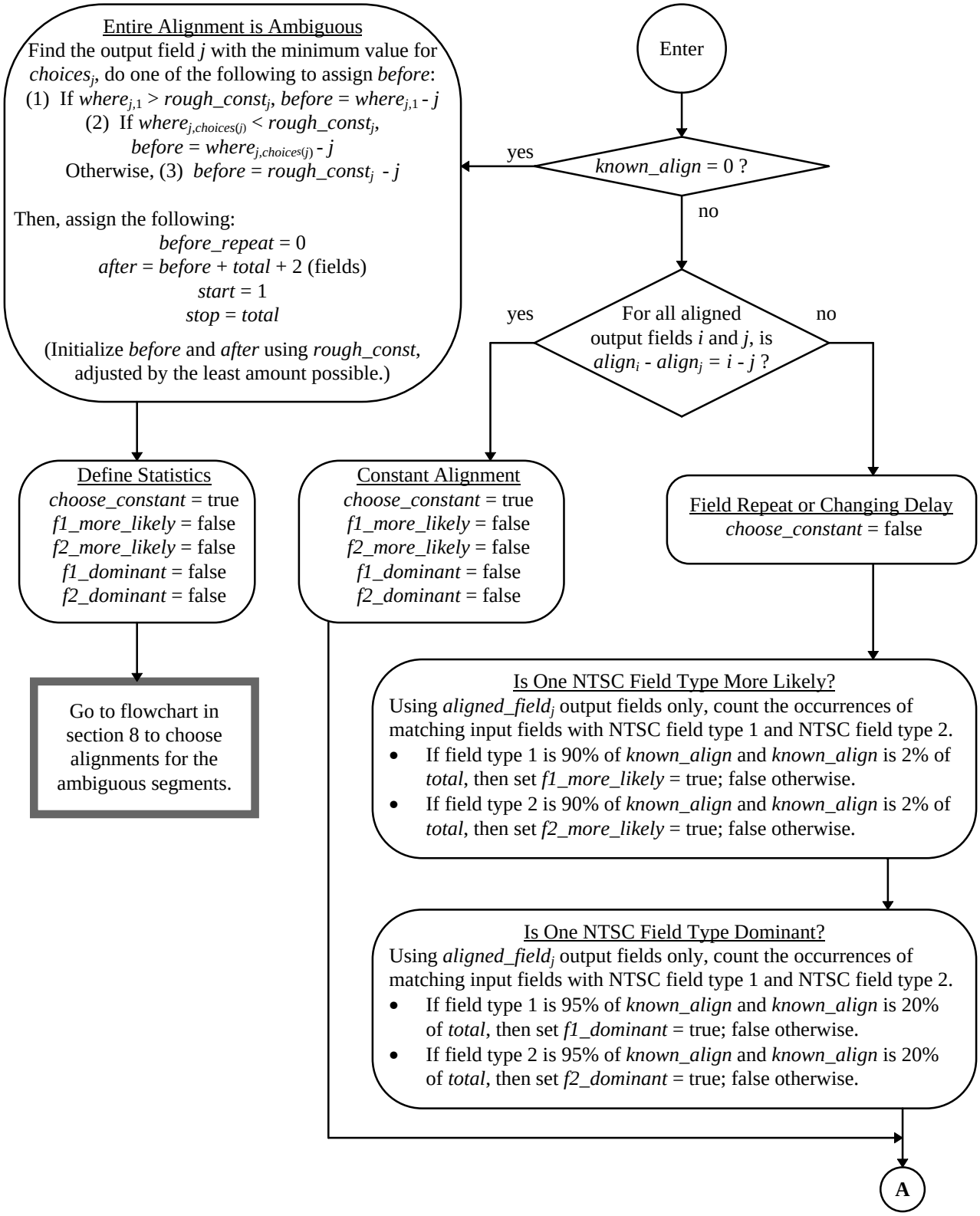

Figure 8. Identify field alignment patterns.

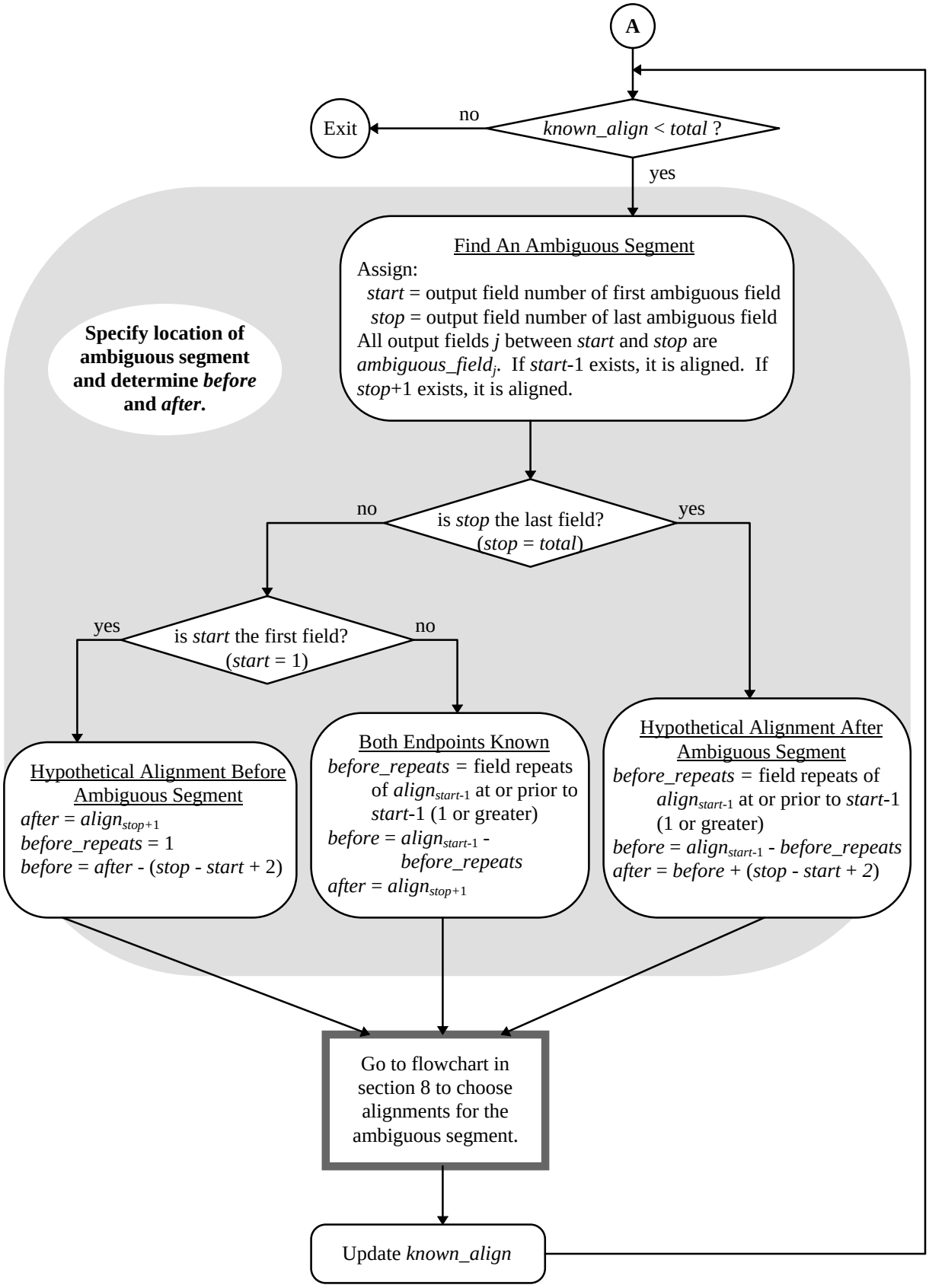

Figure 9. Locate ambiguous fields and resolve ambiguities.

8. Choose Alignments for an Ambiguous Segment

This section presents a method for utilizing the update patterns and statistics found by the algorithms in sections 6, 7 and 8 to choose plausible alignments for the ambiguous segments. The artificial intelligence algorithm presented here takes the patterns identified in the unambiguously aligned output fields and attempts to replicate them on those output fields for which alignment is still ambiguous. This algorithm may be loosely defined as following a constant alignment through the ambiguities, while attempting to impose a field update pattern similar to that seen in the aligned segments. Whenever possible, chosen field alignments either repeat the previous field's alignment, or go to a new future alignment. Backward jumps in alignment are avoided whenever possible.

8.1 Definitions

update_rate is determined by the algorithms in section 4.

The following variables are determined by the algorithms in section 7:

$align_j$

before and *after*

before_repeat

$choices_j$

choose_constant

f1_dominant

f2_dominant

f1_more_likely

f2_more_likely

start and *stop*

$where_{j,i}$

Other Variables Used Within These Flowcharts

$before_const_j$ is the extrapolated constant alignment for output field *j*, presuming constant alignment starting at the alignment of output field (*start* -1) to input field *before*. This is the target constant alignment that the algorithm will try to approximate.

$have_repeated_j$ is the number of contiguous output fields up to and including output field *j* that all align to the same input field. For example, if $have_repeated_j = 3$, then output fields *j*, *j*-1, and *j*-2 all align to the same input field.

j is a loop variable from *start* to *stop* that is used to examine each output field in the ambiguous segment.

x is an offset within $where_{j,x}$ that specifies an input field under consideration when choosing a new alignment, $1 \le x \le choices_j$.

8.2 Ambiguity Resolution Algorithm Flowchart

Figure 10, Figure 11, and Figure 12 contain the overall flowchart which describes the ambiguity resolution algorithm.

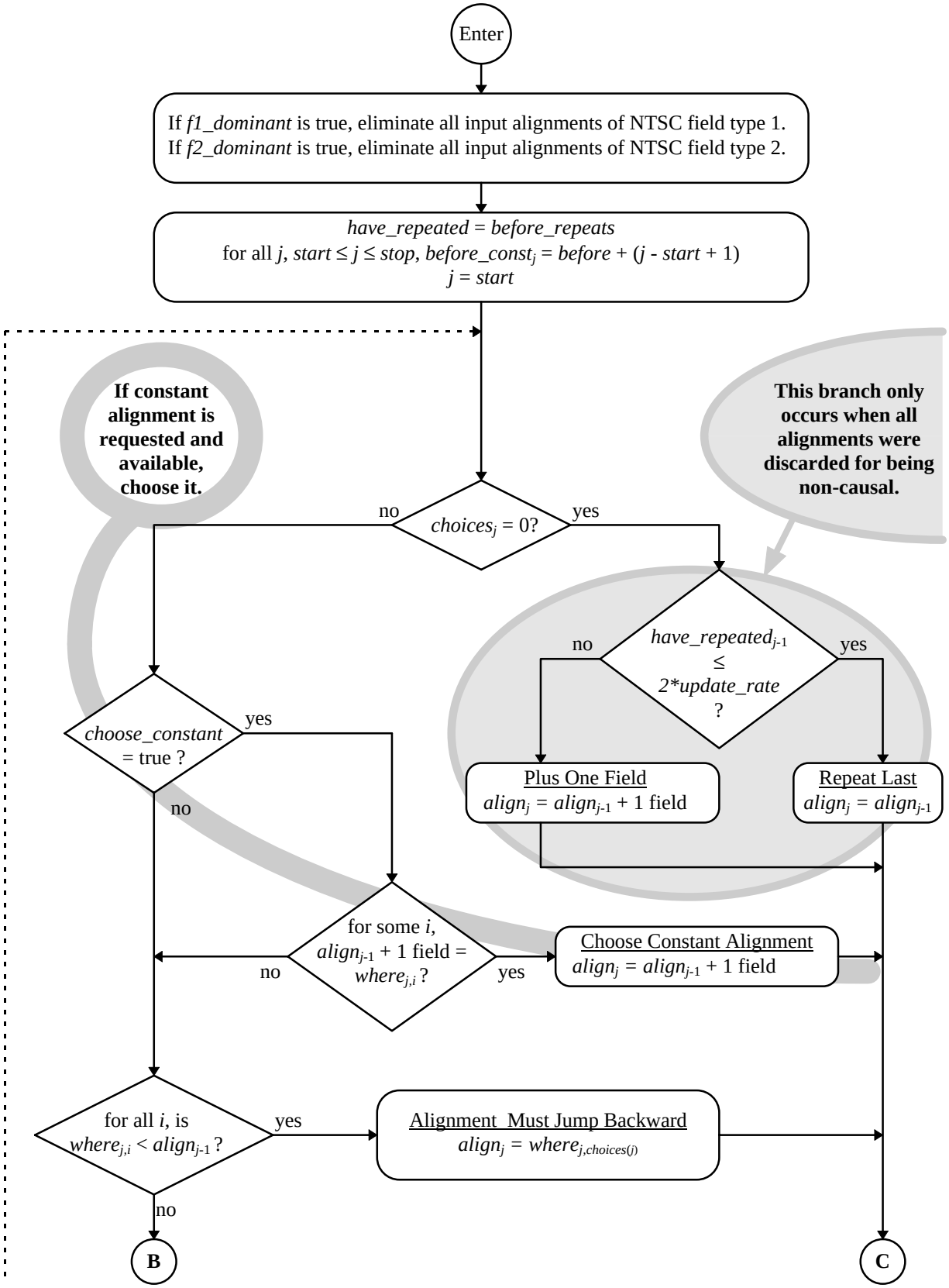

Figure 10. Ambiguity resolution algorithm.

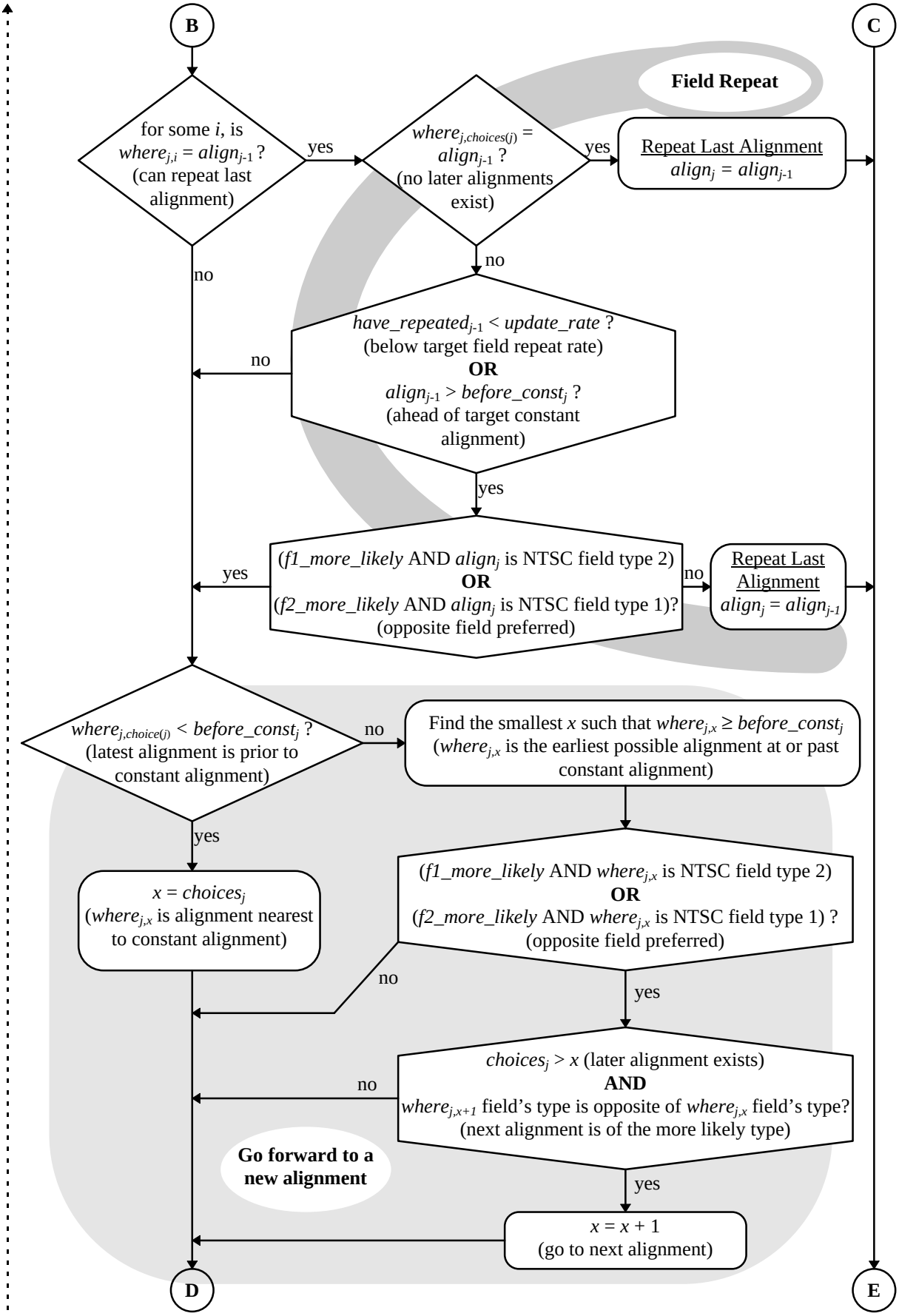

Figure 11. Ambiguity resolution algorithm (continued).

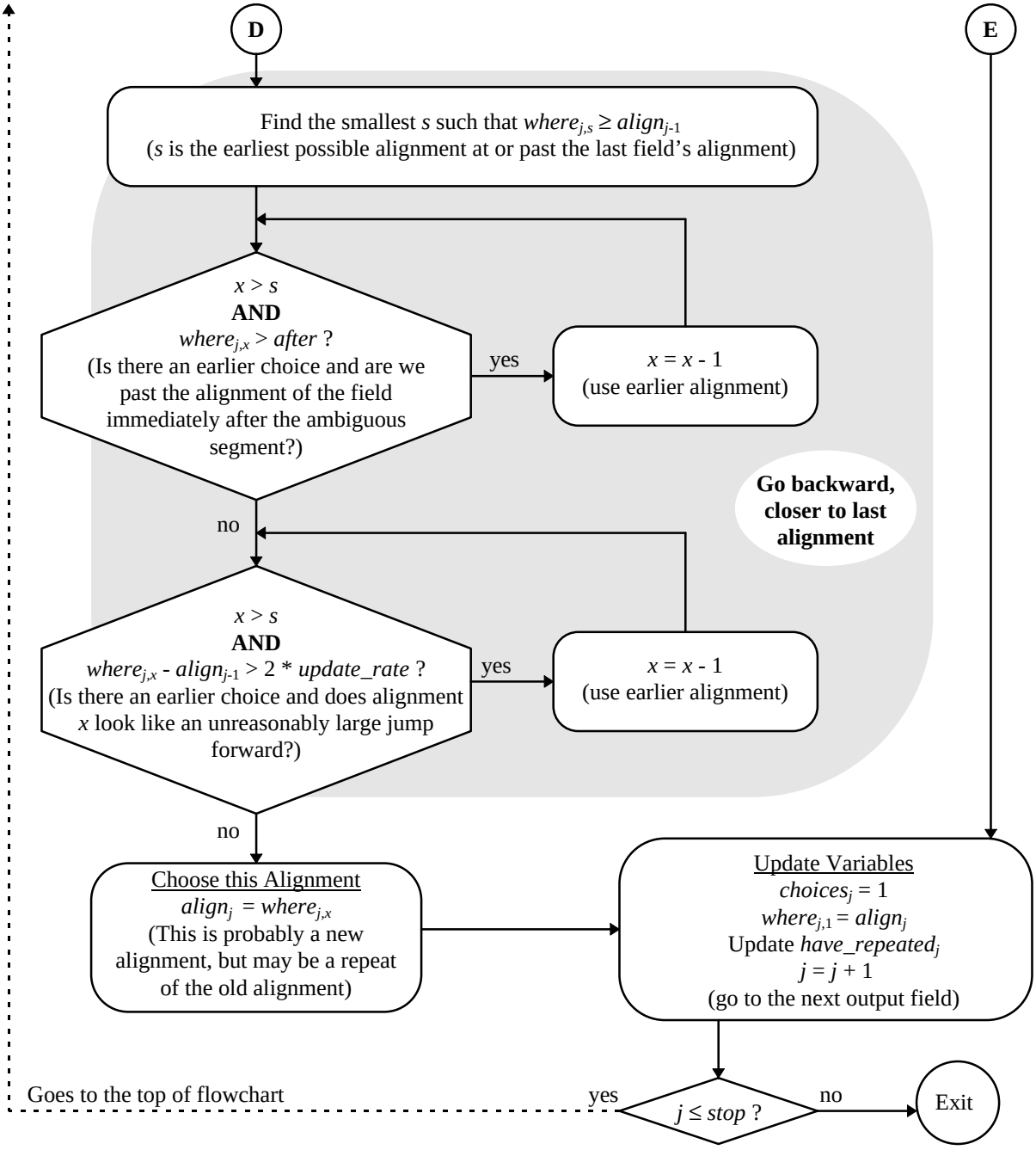

Figure 12. Ambiguity resolution algorithm (continued).

9. Results and Conclusions

Preliminary testing of the alignment algorithms was performed using video clips from the T1A1 data set [5]. This data set contains a wide range of scenes and compression systems. Output fields and their corresponding input fields (as determined by the alignment algorithm given here)

were sent to a video disk recorder and recorded in sequential order (output field 1, input field corresponding to output field 1, output field 2, input field corresponding to output field 2, etc.) so that human examiners could step through the fields and precisely note any misalignments.

As expected, the amount of motion in the scene was found to be the single largest contributing factor for successful field alignment. For instance, the scene vtc2mp (a full profile shot of a woman talking) contains so little motion that the artificial intelligence algorithm (sections 6 to 8) is called for virtually all of the output fields. Conversely, output fields from the scene ftball (a clip from a football game including a thrown football and a tackle) can be entirely aligned using just the signal processing part of the algorithm (sections 3 to 5). Alignment mistakes from the signal processing part of the algorithm are rare. When the artificial intelligence algorithm is called because alignments remain ambiguous after section 5, usually the correct alignment or an alignment very close to the correct alignment is chosen.

When field updates and repeated output fields are present and correctly identified (section 4), cross-checking of this information with the correlation alignment data (section 5) greatly increases the reliability of alignment estimates. Unfortunately, the selection of the optimal motion threshold that can be used to separate field updates from repeated fields is scene dependent and no automated method has yet been found that can be used to set this optimal motion detection threshold.

In conclusion, this contribution has presented variable alignment techniques that work well with subsampled image data. This makes it possible to implement real-time, in-service video delay measurements that can be used to estimate the video delay of individual output fields. All of the techniques presented in this contribution can also be applied to whole field video delay algorithms such as those given in ANSI T1.801.04.

10. References

[1] ANSI T1.801.04-1997, “American National Standard for Telecommunications – Multimedia Communications Delay, Synchronization, and Frame Rate Measurement,” American National Standards Institute, 11 West 42nd Street, New York, New York 10036.

[2] ANSI T1.801.03-1996, “American National Standard for Telecommunications – Digital Transport of One-Way Video Signals – Parameters for Objective Performance Assessment,” American National Standards Institute, 11 West 42nd Street, New York, New York 10036.

[3] Margaret Pinson and Stephen Wolf, “Low-Bandwidth Techniques for Estimating Temporal Delays Between Input and Output Video Sequences,” ANSI contribution T1A1.5/99-204, May, 1999.

[4] Stephen Voran and Stephen Wolf, "Motion-Still Segmentation Algorithm for VTC/VT Objective Quality Assessment," ANSI contribution number T1Q1.5/91-110, January 22, 1991.

[5] A. C. Morton, “Subjective Test Plan (Tenth and Final Draft),” ANSI T1A1 Contribution T1A1.5/94-118R1, October 3, 1993.