DELPHI Collaboration

DELPHI 98-163 PROG 236 TRACK 92
LC-Tool-1999-007 TESLA
19th October 1998

A New Algorithm For Solving Tracking Ambiguities

Daniel Wicke
Bergische Universität-GH Wuppertal

Abstract

The reprocessing of LEP1 data, for which the VD was incorporated into the central tracking algorithms, lead to a reduction of systematic tracking errors on precision measurements involving heavy quarks. In complex event topologies, like b-quark or D^*-events, the large number of ambiguities in VD hit association makes the ambiguity resolution a key point within the improved tracking algorithms. The basic ideas, the implementation and the properties of a new algorithm developed for the purpose of solving such tracking ambiguities are described.

1 Introduction

In 1995 it became clear that many precision measurements based on the LEP1 data were limited by tracking systematics. Especially R_b and other measurements involving reconstruction of secondary vertices suffered from problems in the vertex detector association in charged tracks. Therefore the incorporation of the vertex detector into the central tracking algorithms was the major goal of the new pattern recognition code for charged tracks in the central event reconstruction program called DELANA[1]. This project lead to a final reprocessing of the full LEP1 dataset and subsequently to an enormous improvement of many precision measurements.

In the new pattern recognition the vertex detector hits are used to steer the track finding in addition to the standard searches based on the TPC. In dense tracking situations like b- or c-quark jets many ambiguous hit combinations are generated. To preserve the optimal hit association to the tracks the solution of track ambiguities is of central importance.

The increasing number of tracking ambiguities and the new event topologies arising through the inclusion of the vertex detector lead to new challenges for the ambiguity processor. The algorithm developed to meet these new challenges is described in this note.

First an overview over the programming environment inside the event reconstruction program DELANA will be given to define some technical terms. Then in Section 3 the algorithm used before the LEP1 reprocessing is described. The new algorithm and its implementation is presented in Section 4. Special ingredients for both algorithms are the allowed sub-track detector combinations and the scoring system described in Section 5 and 6. Problems and possible improvement are discussed in Section 7.

2 DELANA

To prepare the data taken by the DELPHI detector for physics analysis, the raw data as recorded by the individual DELPHI detectors are processed by an event reconstruction program named DELANA[1]. DELANA applies detector calibrations, performs track finding and finally calculates particle track parameters.

For this note the tracking part of DELANA is of central importance and thus shall be briefly explained here: First the raw data as stored by the online system are corrected with calibration parameters for each detector component. On these so called TDs, an individual local pattern recognition for each detector component is performed to create (short) track elements called TEs. These first two steps are performed by separate codes for each component.

From the TEs a global pattern recognition called track search is started. With different algorithms (historically divided into barrel and forward track search) TEs are combined to reconstruct particle tracks using all available information. These track candidates stored as TSs are fed into the track fit which determines the precise track parameters. The fit eventually removes TEs detected as outlayer from the TS, and may even reject the complete TS, if the χ^2 is too bad. The remaining TSs are then stored in structures called deactive TKs.

So far no precaution has been taken to assure that individual measurements are not used multiple times. The underlying system called TANAGRA[2, 3] into which the different structures are stored takes care of the bookkeeping of this information though. Before the TKs can be finally stored, the possible double uses have to be removed. On the level of TKs this implies the selection of a consistent set of TKs and to reject the others as ghosts. This is the task of the ambiguity processor. The solution found by the ambiguity processor is stored by "activating" the corresponding TKs.

In order to give the detector component modules the opportunity to add further data to active TKs, that would not be reliable on their own, a second stage is performed. For each detector component a 2nd stage local pattern recognition is called, which may add new TEs to existing TKs or even create new stand-alone TEs using the information of already agreed active TKs. Afterwards the track search, track fit and the ambiguity processor are run again on the changed and new informations.

Active TKs are further used to setup a vertex structure, they are also needed for the RICHes to make their mass assignments. The results of DELANA are finally stored to tape in a format called Full DST.

In the following sections the technical names TE, TS and TK will be used for preciseness. The reader may read these as track element (TE), track candidate (TS) or track (TK) though.

3 The Old Algorithm

As described above the ambiguity processor has to decide which of the track candidates supplied by the track searches and successfully fitted by the track fit (deactive TKs) will be accepted for physics analysis. As the track searches may suggest several TSs making use of one and the same TE, not all TKs may be accepted at the same time.

I will call two TKs mutually excluded when they have at least one TE in common (or if they use two TEs which were defined to be mutually excluded themselves). A set of TKs in which no pair of TKs is mutually excluded will be called consistent. In these terms, the task of an ambiguity processor is to find a set of consistent TKs.

The old ambiguity processor operates in two steps. In the first step it chooses such a set of consistent TKs by selecting TKs from the set of supplied TKs while maximising the number of accepted TKs and the number of TEs used. To prepare the second step from all TKs remaining, TEs are removed thereby building shortened TKs such that these (shortened) TKs are no longer mutually excluded to any of the selected set. The detector contribution of the shortened TKs is checked and only combinations of known good quality are kept. Now step one is repeated on these selected modified TKs. The second iteration is often called the recovery stage.

As long as tracks were mainly defined by the TPC in the barrel or FCA+FCB in the forward region this method had no problems, because tracks without these main tracking modules could not exist or were of poor quality. Thus the recovery phase is only used very seldomly.

The introduction of a micro-vertex detector changed this situation in several aspects. More and more tracks needed to be recovered as the development of track search methods including the VD carried on. Possibly tracks were even lost due to the fact that the recovery step wasn't repeated. Even more important is that the quality of the tracks recovered could in no way influence the decision made in the first step. In this sense the decisions were taken using only parts of the information available about the event.

It became possible to create tracks of good quality without the central TPC. Second the association of VD points to a track very often is ambiguous, as the extrapolation bands from outside tracks often overlap. Also one extrapolation band may fit to several separated vertex detector hits. These latter aspects enlarged the number of ambiguities given by the track searches and created more complex topological situations, so that more tracks entered the recovery phase.

During the incorporation of the vertex detector into the track search it became clear that choosing the best set of consistent tracks from a fixed set of tracks was a major draw back. First it was tried to solve this problem by including shortened TSs of the found tracks into the track search results, thereby enlarging the fixed set such that all potential TKs are included. This caused problems due to large amount of CPU needed by the ambiguity processor to solve this. In turn a preselection of tracks was introduced into the barrel track search anticipating parts of the tasks of the ambiguity processor. Besides the problem of having to make the decision on basis of a partially solved event, due to internal differences of the track search methods the preselection was not applied to the results of all search methods, thus leading to an inconsistent treatment of the results.

For these reasons it was finally agreed to rewrite the ambiguity processor.

4 The New Algorithm

With the new algorithm the old clear separation of tasks should again be assured.

- The track searches shall find candidates of tracks. These candidates should contain as many detector components as possible.
- The fit shall find the optimal parameters describing these tracks. It is allowed to remove outlayers in order to assure a good quality of tracks.
- The ambiguity processor now shall find the best set of consistent tracks out of the given tracks.

The solution given by the ambiguity processor shall take into account the tracks that will occur in a recovery like procedure for its decision, i.e. it shall consider all possible sub-TKs of a given TK. For this reason the new algorithm will treat the given tracks no longer as fixed objects, but is allowed to remove TEs when necessary.

4.1 Main Ideas

The idea of a step in the old ambiguity processor was to select a *set* of TKs. In the new algorithm this set becomes an *ordered list*. The order determines from which TK one or more TEs are removed in order to keep the solution consistent. This is done by requiring that a TK may not use TEs that are already used by a TK further to the top of the list. If a TK containing TEs that are already used shall be entered into the list, the already used TEs are simply removed from this TK.

With this idea sub-TKs (i.e. TKs containing a subset of TEs from some other TK) enter exactly where they are needed. They need no longer be given to the ambiguity processor, so that the number of TKs to handle reduces.

So the base algorithm goes as follows:

- Put one TK on the list.
- Remove all used TEs from the remaining TKs.
- Remove disallowed detector combinations.
- Refit track parameters of all shortened TKs.
- Repeat these steps until no more TKs are remaining.
- Continue creating lists until all possible lists are checked.
- Return the list with the best score as solution.

In order to find the optimal solution of n candidates eventually $n!$ possibilities need to be tested. An algorithm testing all these $n!$ possibilities will not be able to solve more than 10 to 12 tracks within a reasonable amount of time. Thus a large effort was made to reduce the number of situations to be tested, by removing identical solutions from the testing-list.

4.2 Reducing the number of combinations to be tested

The ordered list of TKs to be tested is constructed from top to bottom. First a TK is entered into the list, then all used TEs are removed from the remaining TKs. Then the next TK is entered and so on.

During the removal of TEs the shortened TK produced may be sub-TKs of other TKs in the remaining set. Also identical TKs may appear from different origins. So removing sub-TKs and doubles will reduce the number of TKs to handle without changing the result. Sub-TKs and doubles are removed on entry to the ambiguity processor as well, as they may also have appeared from different track search methods.

An even more powerful method of reducing the number of combinations to be tested uses the fact, that the order of two tracks is only relevant, when the two tracks are mutually excluded. Before starting to test different combinations, the tracks are therefore sorted into sets: The first set starts with the first track. All tracks mutually excluded with the first one are added to this set. Thereafter all tracks mutually excluded to any of the newly added tracks are entered into the set. This is repeated until no more tracks were added. Then a second set is started with the first track not in the previous set(s). Further sets are created until each track is associated to a set. Tracks that will be sorted in one and the same set with the above procedure are called connected tracks.

Now each of the sets is solved separately. As these sets may fall apart when one track is removed in order to be put on the solution list, it is useful to apply the separation of tracks into sets each time a track is put on the solution list.

This leads to the following refined recursive algorithm:

A
- Remove all sub-TKs and double TKs and TKs of disallowed detector combinations from the incoming tracks.
- Separate the TKs into sets of connected TKs.
- Solve each set with algorithm B.

- Join the solutions of the sets to one list and return it as solution.

B
- Loop over all TKs i
- Calculate the score for the solution list, that starts with TK i and is followed by the solution of the remaining (appropriately shortened) TKs as returned by calling A with these remaining TKs.
- Return the solution with the best score.

This recursive algorithm gives the same solution as the base algorithm given above, but needs to test orders of magnitude less combinations. (See example in Fig. 1)

4.3 Further Optimisations

In the course of this algorithm sub-TKs of a given TK are computed very often. Thus it is useful to store the results of all time consuming operations. This includes the result of the track fit and the corresponding χ^2 probability. To access the sub-TK of a TK with a given TE removed, a pointer to this sub-TK is stored within the local (extended) TK structure. Only if the pointer is NULL, the sub-TK has to be calculated. As identical sub-TKs can occur from different sources, one can check for existence after constructing the sub-TK. This is realised with a global list of TKs via a hash number associated to each TK. Only if two (sub-)TKs have an identical hash number a detailed comparison is needed. If a substring is not yet in the global TK list, the parameters are refitted. Sub-TKs that are fitted successfully are then added to the list of TKs. When the fit fails the constructed sub-TK is set to be empty.

To further improve the performance of the algorithm, a function to calculate an upper bound for the score that can be reached with a given set has been implemented. Using this function, the recursion can be stopped, when the score needed to reach a new best solution cannot be provided by the remaining set of tracks.

4.4 Fall-back Algorithm

Even though the methods described so far reduce the number of combinations to be tested by some orders of magnitude and minimise the amount of operations needed to calculate these combinations, one still has to be prepared to deal with events that cannot be exhaustively tested with the above algorithm. In this case the solution is created with a simple algorithm that takes the TK with the highest score from the set to be the first of the list, then removes used TEs from the remaining TKs and then takes the best TK of the remaining set to be the second TK, and so on. This algorithm needs $\mathcal{O}(n^2)$ steps to construct a solution for n TKs.

In order to decide when to use this fall-back algorithm, at first the CPU time was measured and a timeout parameter defined when to give up with an event. To allow the algorithm to be run on different machines giving up on the same events, later a counter was implemented which was tuned to be roughly proportional to the CPU time needed[1]. For the most hopeless cases where the number of TKs in a connected set was too large, the logic gives up immediately, by restricting the recursion depth.

[1] The CPU base timeout was only used for the 94C DATA reprocessing.

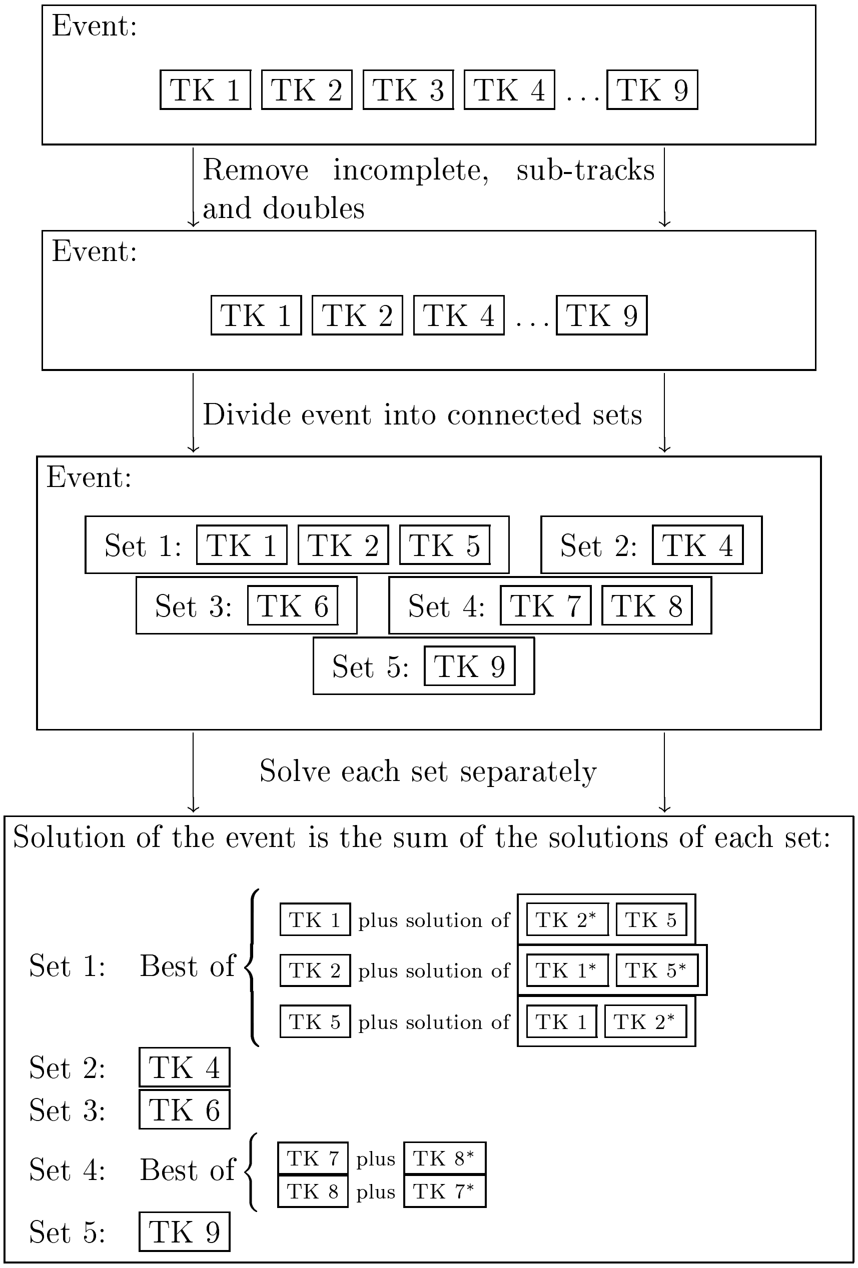

Figure 1: Schematic view of the recursive algorithm to solve ambiguities. From an incoming event of 9 TKs, that may use one and the same TE in different TKs, sub-TKs and doubles are removed (TK 3 disappears). The event is then divided into connected sets. Each of the sets is solved separately. For sets with only one TK (Sets 2, 3 and 5) this TK is given back as the sets solution. For sets with $n > 1$ TKs (Sets 1 and 4), each of the TKs is put on trial to the top of the solution list. The remaining TKs get all TEs removed (denoted by *), that are already used by this first TK and are refitted with the track fit. The solution of this remaining subset is determined by treating it as if it was an event on its own. The solution for the complete event is the sum of the solutions for each set. An example event with the properties used in this description is given in Fig. 2.

LEP1 reprocessing (Version 2.36)	LEP2 tracking including the VFT (Version 2.48)
VD	VD
ID	ID
--	TPC (when short or special forward TE)
OD	OD
FCA	FCA
RIF	RIF
FCB	FCB
ID, VD	ID, VD
OD, VD	OD, VD
OD, ID, VD	ID, OD, VD
FCA, VD	FCA, VD
RIF, VD	RIF, VD
FCB, VD	FCB, VD
ID, OD	ID, OD
ID, FCA	ID, FCA, n(VD)<=1
ID, RIF	ID, RIF, n(VD)<=1
ID, FCB	ID, FCB, n(VD)<=1
FCA, RIF	FCA, RIF, n(VD)<=1
RIF, FCB	RIF, FCB, n(VD)<=1

Table 1: List of forbidden detector combinations. VFT is considered to be part of the VD.

5 Allowed sub-TKs and detector combinations

When removing TEs from a TK, the remaining TEs may define the track parameters of the TK only poorly or not at all. In this case no sub-TK should be built, but the TK should be removed from the set of remaining TKs. To define this situation a list of forbidden detector combinations is defined. The lists used for the LEP1 reprocessing and the one used with the new VFT tracking are shown in Table 1. In the old algorithm a corresponding list was used to decide which of the shortened TKs, were to be put into the recovery phase.

Further it was found that it is useful to forbid the use of a sub-TK A^* of TK A when A^* is created by removing TEs used in TK B and the sub-TK B^* is invalid. B^* is the sub-TK that appears when removing from B the TEs used by A. This reduces the number of unwanted split tracks[2].

[2]This split protection was only developed after the reprocessing of LEP1 data was already started and is thus only used for processings of 1996 and later data sets.

Element	Score
`VD`	10
`ID`	3
`TPC`	20
`TPC_FORW`	3
`OD`	3
`FCA`	3
`RIF`	3
`FCB`	3
`TK fit`	$100 + \log P(\chi^2, \mathrm{ndf})$

Table 2: Scoring system: The score of a given solution is build as the sum of the scores of the elements used. `TPC_FORW` TEs are specially marked (mostly forward) TPC TEs with reduced quality. These are distinct from `TPC` TEs only after the LEP1 reprocessing.

6 Different Scoring Systems

For each possible solution a *score* is calculated and the solution with the highest score is taken as the final solution. There are many possible scoring systems. A reasonable scoring should incorporate the result of the track fit, i.e. the probability of a certain TK to be the true solution. It should favour solutions using more TEs than others. And finally one needs to investigate, whether to minimise or to maximise the number of TKs in an event.

For ease in programming and for speed, the score of an event was chosen to be the sum of the scores of the TKs building up this event.

The fit result is most naturally included via the χ^2-probability of the fit result. To obtain an additive scoring system, $\log P(\chi^2, \mathrm{ndf})$ is used. The number of TEs is maximised, by giving an additional score for each TE used in a TK. These TE scores must not depend on the TK the TE is used in, so that the decision into which TK a TE is put, will be made on grounds of the fit results. TEs of different detectors were given different scores, such that the more important detectors would not be discarded in favour of less precise ones. The investigations regarding the scoring systems showed that it is better to maximise the number of TK, thus a base score of 100 per TK was introduced.

The scoring function is summarised in Table 2

7 Possible Improvements

Solving the event by ordering the list of found TKs, allowed the recursive and elegant implementation of a fast ambiguity processor. But this ansatz results in the major drawback of this algorithm for ambiguity processing.

Assume you have two TKs competing for two TEs. By ordering the TKs one will put both TEs in one and the same TK (e.g. like TK1 and TK2 in Fig. 2 competing for VD2 and VD3). The possibilities of putting one TE into TK1 an the second TE into TK2 or vice versa are not considered. Given the importance of letting the probability

TK 1:	VD 1	VD 2	VD 3	ID 1	TPC 1
TK 2:	VD 2	VD 3	ID 2	TPC 2	
TK 3:	FCA 1	FCB 1			
TK 4:	VFT 1	FCA 1	FCB1		
TK 5:	VD 4	VD 5	ID 2	TPC 2	OD 1
TK 6:	VFT 2	ID 3	FCA 2	FCB 2	
TK 7:	VD 6	VD 7	ID 4	OD 2	
TK 8:	VD 8	VD 9	ID 4*	OD 3	
TK 9:	VFT 3	ID 5	TPC 3	FCB 3	

Figure 2: An example event that may enter the ambiguity processor with the set of deactive TKs given by their TE contents. ID 4 and ID 4* are two TE's describing ID hits and their mirrors. This event serves as an example for Fig. 1.

decide where ever possible, this may hold an improvement of another 2% in numbers of reconstructed D^*, when it would be possible to implement this without loss of speed.

A second but not as important drawback is that due to the recursive nature of the algorithm preliminary results are not available during the calculation. In case of a timeout situation, this results in the total loss of CPU time used for solving this event so far, restarting the analysis from scratch with the fall-back algorithm.

These shortcomings could be addressed in an algorithm, that allocates the TEs being associated to different TKs to one of the TKs after the other. In this case the order in which one decides to which TK a special TE belongs is of no interest, so that an iterative loop should be possible for this job. Ordering the loop appropriately should make it possible to start with a solution of the "n^2-type" quality, improving it in further steps.

This way an outstanding property of the old ambiguity processor could be reassembled again, namely the ability to return more than a reasonable solution at any given time. A fall-back algorithm would no longer be needed.

8 Acknowledgements

I would like to thank M. Elsing and J. Wickens for their discussions, suggestions and support during the development of the code.

References

[1] DELPHI Collaboration. Delphi data analysis program (delana) user's guide. DELPHI 89-44 PROG 137, 1989.

[2] D. Bertrand and L. Pape. Tanagra: Track analysis and graphics package. DELPHI 86-75 PROG 55, 1986.

[3] D. Bertrand and L. Pape. Tanagra, track analysis and graphics package. DELPHI 87-95 PROG 98, 1997.