Parallel Multiple Key Sort Algorithm for Traverse Data Set

C. P. E. Agbachi

Departmentof Mathematical Sciences, Kogi State University

Anyigba, Kogi State, Nigeria

Abstract*— *Sorting traditionally, is based on a single key algorithm. Even in multi-key situations, the mode of operation centres on one key at a time. Outside this model, the procedure is unsatisfactory making a search for adaptive solution imperative. A case in point is traverse data collection in Geomatics Engineering, where sorting is required for the purpose of reconstructing survey topology. This process has always been fraught, especially in large networks. Thus, this paper discusses the methods and techniques of a successful model sort in synchronised three-key data structures.

Keywords—Total Station, 3-Tripod Setup, Bowditch Adjustment, Objects.

I. INTRODUCTION

A traverse is one of the established methods of surveying for horizontal positions. In the early days, it had to compete with other forms such as triangulation and trilateration, involving angle and distance measurements respectively. With the advent of digital instruments, Total Stations, all the modes have evolved into one category, Traverse. It is thus, with detail descriptions, the mainstay in practice of Surveying Engineering [1, 2].

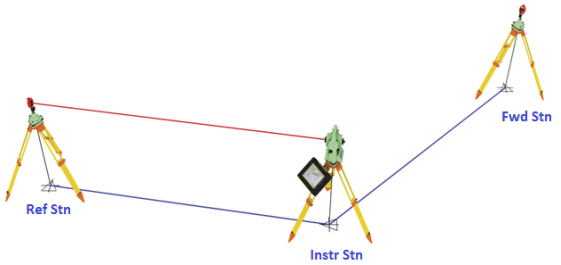

Fig. 1 Traverse Survey

A traverse, Fig 1, is characterised by setups defined by the instrument position, reference station and then the forward station. In what has come to be known as 3-tripod setup, the instrument station moves to the forward station, while the reference station is dismantled and setup in the next forward point. This routine starts from known positions and ends likewise in control points. Computation is mostly by Bowditch Adjustment [3].

In normal circumstances, surveys follow strict order of design in topology. Nevertheless, field work in recent days are dynamic, may involve as many as 100 setups in networks of pipeline construction and even more in larger layouts. Before now, such surveys tend to be computed on patch basis, a procedure that leaves room for inconsistency. Ideally given computing resources, the entire survey can be computed in a single frame of adjustment. However, because the surveys would have been carried out in batches by different groups, such a contiguous data set would be possible only through sorting.

A. Field Model

Sorting in the field arises due to incomplete surveys that are concluded at a later time. A case in point is illustrated below.

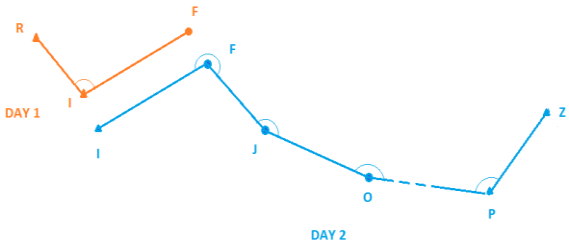

Fig. 2 Two-Day Survey

Imagine an observation at I with respect to stations R and F on day 1. If the work could not continue until the next day, connection is maintained through the subsequent observation. Note that the instrument is now setup at station F with reference to station I.

There could also be a reverse situation where tasks in Day 1 and Day2 are interchanged, such that the bulk of the work is carried out in the first day. Then on the last day, the connection to starting controls I and R is established.

Other situations abound in a large network of traverses, where the constraints of time demand as a priority the completion of survey. That leaves the sorting into survey topology and computability, an undertaking in the office.

B. Sort Algorithm

There are three keys to be considered, in this process. These are:

1. Instrument Station
2. Reference Station
3. Forward Station

Upon examination, every setup is characterised by $S_n = \{R_{n-1}, I_n, F_{n+1}\}$ where n is the station number in the sequence. R_{n-1} is the previous or reference target station and I_n is the instrument station. Likewise F_{n+1} is the next or forward target station.

For every S_n and $S_{n+1} = \{R_n, I_{n+1}, F_{n+2}\}$, it can be shown [4, 5] that $S_n \cap S_{n+1} = \{I, F\}$ --- (1)

Similarly, $S_n \cap S_{n-1} = \{R, I\}$ --- (2)

Thus, Sort algorithm requires parallel evaluation of three keys to determine a location for insertion.

1. *Forward Search:*

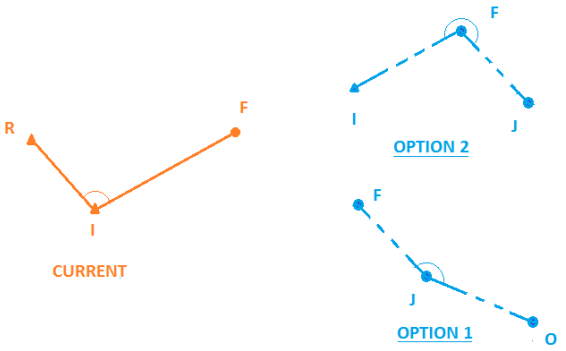

Fig 3

The forward search aims to append an observation to the current instrument position, Fig. 3, in line with survey topology.

Taking the current location $S_n = \{R, I, F\}$, and option 1, $S_{n+1} = \{F, J, O\}$ it is clear that $S_n \cap S_{n+1} = \{F\}$. Therefore there is no continuity by this option.

On the other hand, for $S_{n+1} = \{I, F, J\}$, $S_n \cap S_{n+1} = \{I, F\}$. Hence, the connection exists. Further details maybe expressed as follows:

At the current location, the next setup meets the following conditions:

1. Instrument Station is the Reference Station at the next setup.
2. Forward Station is the Instrument Station at the next setup.

2. *Reverse Search:*

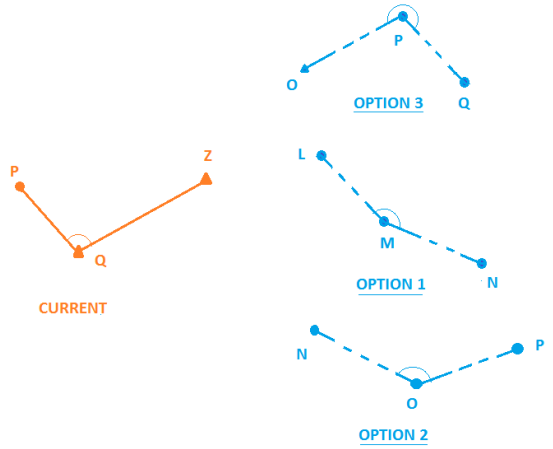

Fig 4

Reverse search arises in order to insert a setup before the current location, Fig. 4, in the topology. Thus, considering $S_n = \{P, Q, Z\}$ and option 1, $S_{n-1} = \{L, M, N\}$, $S_n \cap S_{n-1} = \{\Phi\}$. For option 2, $S_n \cap S_{n-1} = \{P\}$. However in option 3,$S_n \cap S_{n-1} = \{P, Q\}$. Hence, the process leads to required solution. Further details may also be expressed as follows:

At the current location, the previous setup meets the following conditions:

1. Instrument Station is the Forward Station in the preceding setup.
2. Reference Station is the Instrument Station in the preceding setup.

II. Data Structures

Data Structure is a model of organisation and information description [6]. In this respect, the task is modelling of a setup with a view to accomplish the Sort Algorithm. Towards this, there are two object structures to consider. The first is the description of observations at the instrument station. The second is object representation of the field book, a database, to which surveyors book, reduce and sort observations.

A. Setup

SetUp Frame

Data1	Reference Station
Data2	Instrument Station
Data3	Forward Station
Data4	:
	:
Method	:

Table 1

Setup is based on Knowledge Representation and starts with a conceptual design, Table 1, in form of a Frame [7, 8]. This is characterised by slots or attributes and methods. There are thus slots for Ref Stn, InstrStn and Fwd Stn. Also included are measurements, directions, angles and distances. Then the Methods are procedures that manipulate data for desired result. For instance, within the Frame, methods reduce the observed angles and perform corrections to observed slope distances.

```
type
   PRedTravRec = ^TRedTravRec;
   TRedTravRec = record
     StnName,RefStn,FwdStn: array [0..5] of Char;
     InstrHt,TgHtRO,TgHtFwd: array [0..5] of Char;
     AngleRO,AngleFwd,ElevRO,ElevFwd: array[0..15] of Char;
       .
       .
       .
     Easting,Northing,Height: array [0..15] of Char;
     EastingRO,NorthingRO,HeightRO: array [0..15] of Char;
     EastingFwd,NorthingFwd,HeightFwd: array [0..15] of Char;
   end;

   PRedTravObj = ^TRedTravObj;
   TRedTravObj = object(TObject)
     TravRecord:TRedTravRec;
     constructorInit(Fill:Boolean);
     constructor Load(var S:TStream);
     procedure Store(var S:TStream);
    destructor Done; virtual;
  end;
```

Fig. 5

Frames translate into programmable object representation, Fig. 5, through a supporting language. A typical example is Pascal as described in [9, 10].

B. Field Book Container

Field book is a container or database object that holds the observations. It is analogous to actual field book in that it stores the observations. Furthermore, sorting usually performed manually in the field book is carried out by methods in the database object.

A comprehensive discussion and description of the container can be found in [11]. Of note, is the room for adaptation to use in Traverse. Hence the field book object has the form as described below.

```
   PTravFieldBook = ^TTravFieldBook;
   TTravFieldBook = object(TCollection)
     function Found(Item: Pointer): Boolean;  virtual;
     functionSearchForPosition(Item: Pointer): Integer; virtual;
     procedure Filter(Item: Pointer); virtual;
     procedure Insert(Item:  Pointer); virtual;
     procedure Error(Code, Info: Integer); virtual;
   end;
```

Fig 6

In this, of significance are the methods, SearchForPosition, and Insert. By this arrangement, observations can be downloaded in any order and sorted into topology of Traverse, Fig. 7.

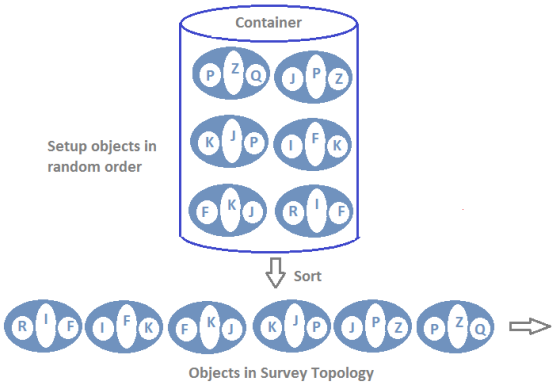

Fig. 7 Triple-Key Sort

III. IMPLEMENTATION

Implementation centres on the operations of key procedures, Filter, Found, Insert and SearchForPosition.

A. Filter

```
procedureTTravFieldBook.Filter(Item: Pointer);
begin
    if (count > 0) or TravFileMode then
  begin
     Reject :=NotValid(Item) or Found(Item);
  end;
end;
```

Fig. 8

The aim of this routine is to avoid erroneous fields in the data set. Whereas such validation may have been performed during data capture, it is always better, from experience, to guard further against surreptitious input. So it is not conceivable that key fields would have identical names. Nor could the data be valid if the fields are empty.

Any other important checks can also be performed in this routine to determine acceptance or otherwise of the data set. For instance duplicate entries are not required.

Filter returns the result of processing functions NotValid and Found in Reject, a Boolean variable, Fig 8.

B. Found

```
functionTTravFieldBook.Found(Item:Pointer): Boolean;
var SearchKey,A1,A2, A3: string;  F:PRedTravObj;

  functionItExists(OldRec:PRedTravObj): Boolean; far;
  var A,B,C :String;
  begin
    A :=StrPas(OldRec^.TravRecord.RefStn);
    B :=StrPas(OldRec^.TravRecord.StnName);
    C :=StrPas(OldRec^.TravRecord.FwdStn);
    ItExists := ((A + B + C) = SearchKey);
  end;
```

```
begin
  A1 := StrPas(PRedTravObj(Item)^.TravRecord.RefStn);
  A2 := StrPas(PRedTravObj(Item)^.TravRecord.StnName);
  A2 := StrPas(PRedTravObj(Item)^.TravRecord.FwdStn);
  SearchKey := A1 + A2 +A3;  F := FirstThat(@ItExists);

  if (F <> nil) then
  begin
   AtFree(IndexOf(F));
   Reject := False;
  end;
  Found := False;
end;
```

Fig 9

Found performs a search for duplicates in the database, during the filtering process. Every set up in traverse survey is uniquely identified by the reference, instrument and forward stations. Consequently, the search key is a concatenation of the three key fields. With this variable, an iteration takes place in the database searching for any previous entry and if found, is deleted to make way for update in observations.

C. SearchForPosition

```
functionTTravFieldBook.SearchForPosition(Item:     Pointer):
Integer;

var   I : integer ; CurrInStn,CurrROStn,CurrFwdStn:string;
X,V,Y:PRedTravObj;
ANode: Boolean;

    functionMatchFwdReadings: PRedTravObj): Boolean; far;
    varFwd,InStn: string;
    begin
      Fwd :=StrPas(Readings^.TravRecord.FwdStn);
      InStn :=StrPas(Readings^.TravRecord.StnName);
         Match_Up   :=   ((Fwd   =   CurrInStn)and(InStn   =
         CurrROStn));
     end;

    functionMatchRev(Readings: PRedTravObj): Boolean; far;
    varFwd,InStn,ROStn: string;
    begin
       ROStn :=StrPas(Readings^.TravRecord.RefStn);
       InStn :=StrPas(Readings^.TravRecord.StnName);
         Match_Down   :=   ((ROStn   =   CurrInStn)and(InStn   =
         CurrFwdStn));
     end;

begin
      CurrInStn  :=StrPas(PRedTravObj(Item)^.TravRecord.Stn
      Name);
      CurrROStn  :=StrPas(PRedTravObj(Item)^.TravRecord.R
      efStn);
      CurrFwdStn :=StrPas(PRedTravObj(Item)^.TravRecord.F
      wdStn);
      V :=LastThat(@MatchFwd); X := FirstThat(@MatchRev);

      if (V <> nil) then
     begin
      SearchForPosition :=IndexOf(V)+1;
      PRedTravObj(Item)^.TravRecord.Link := True;
      PRedTravObj(At((IndexOf(V))))^.
      TravRecord.Link := True;
     end
     else if (X <> nil) then
     begin
       SearchForPosition :=IndexOf(X);
       PRedTravObj(Item)^.TravRecord.Link := True;
           If (IndexOf(X)) < (count-1) then
       PRedTravObj(At((IndexOf(X)+1)))^.
       TravRecord.Link := True;
     end else
     begin
       SearchForPosition := Count ;
        PRedTravObj(Item)^.TravRecord.Link := False;
     end;
end;
```

Fig. 10

The routine SearchForPosition implements the Forward and Reverse Search algorithms. It does so by examining the conditionality in the relationships between the key fields in any pair of data sets.

The function MatchFwd is a forward search the result of which identifies a location next to the current position, to insert or append an observation. Similarly, the MatchRev searches in reverse and if successful, identifies the current object position as location for insertion. By this action, all other objects will move a position down the line.

D. Insert

```
procedureTTravFieldBook.Insert(Item: Pointer);
begin
   Filter(PRedTravObj(Item));
   If not Reject then
    AtInsert(SearchForPosition(PRedTravObj(Item)),
    PRedTravObj(Item));
end;
```

Fig. 11

The routine, SearchForPosition, returns an integer, Index, to the calling program. Hence an insertion is performed by the last call in the Insert method, in the form of AtInsert(Index, NewObject), Fig. 11.

IV. APPLICATION

Fig. 12

In Fig. 12 is a representation of a typical field situation where survey groups on a large scale work download the day's work to management software.

A typical example in this instance is SMS [12]. Thus as illustrated in Fig. 7, the field book in Fig. 13 is a container that holds observations that may be entered or downloaded in any order.

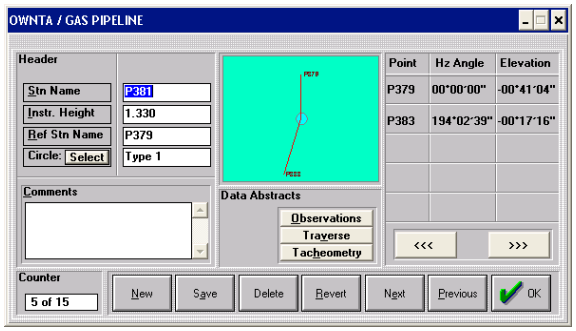

Fig. 13Traverse Field Book

The input process continues until the survey is complete. A contiguous and sorted data set is then generated, Fig. 14, for computation of positions by Least Squares [13] or Bowditch adjustment.

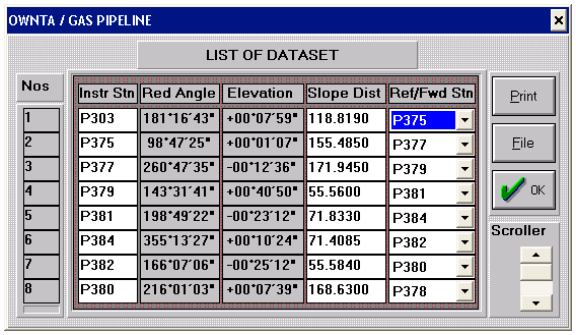

Fig. 14Traverse Data List

V. Conclusions

The model sort described in this paper has proved invaluable in resolving the complexities that arise in dynamic survey data collection. It provides the basis for processing level data sets, especially with the advent of digital levels where data size can be in order of a thousand points. Similarly, it has been adapted for successful application in traverse data collection.

The progression has been from concurrent double key data structures to triple key models. Conclusively then, it is a generic prototype for parallel processing of n-key data structures.

On reflection, the novel solution is due to evolution in programming techniques as conventional approach gives way to object methodology. With this, hitherto difficult situations tend to have model solutions. In this vein, the benefits of customization and in house developments are duly recommended.

References

[1] W. Schofield, M. Breach, "Engineering Surveying",6th Edition, Butterworth-Heinemann, © 2007.

[2] Yuji Murayama Surantha Dassanayake, "Fundamentals of Surveying", University of Tsukuba, Japan.

[3] Department of Civil Engineering, "Surveying - Traverse Calculations", The University of Memphis, www.ce.memphis.edu/1112/notes/project_3/traverse/

[4] Glynn Winskel, "Set Theory for Computer Science", University of Cambridge, © 2010 Glynn Winskel

[5] Jos´eMeseguer, "Set Theory and Algebra in Computer Science", University of Illinois, Urbana,©Jos´eMeseguer, 2008–2012.

[6] Jean-Paul Tremblay, Paul G. Sorenson, "An Introduction to Data Structures With Applications", McGraw Hill Computer Science Series 2nd Edition.

[7] Matthew Huntbach, "Notes on Semantic Nets and Frames", Dept of Computer Science, Queen Mary and Westfield College, London.

[8] C. P. E. Agbachi, "Optimisation of Least Squares Algorithm: A Study of Frame Based Programming Techniques in Horizontal Networks", IJMTT, Volume 37 Number 3 - September 2016.

[9] J.E. Akin, "Object Oriented Programming Concepts", ©2001 J.E Akin, https://www.clear.rice.edu/../oop3.pdf

[10] Marco Cantu, "Object Pascal Handbook", © Marco Cantu 1995-2016. http://www.marcocantu.com/objectpascal

[11] C. P. E. Agbachi, "Design and Application of Concurrent Double Key Survey Data Structures", IJCTT, Volume 36 Number 3 - June 2016.

[12] C. P. E. Agbachi, "Surveying Software", Chartered Institution of Civil Engineering Surveyors ICES, October 2011,http://mag.digitalpc.co.uk/fvx/ces/1110/?pn=44

[13] R. E. Deakin, "Notes on Least Squares 7", Geospatial Science, RMIT University, 2005.