Query Translation Based on Hypergraph Models

M. M. OWRANG O.* AND L. L. MILLER†‡

* *Department of Mathematics, Statistics and Computer Science, American University, Washington, D.C., USA*

† *Department of Computer Science, Iowa State University, Ames, Iowa 50011, USA*

The hypergraph database model is used as the translation model between two relational database designs. Algorithms necessary to do the translation are presented.

Received August 1985, revised December 1986

1. INTRODUCTION

The problem of translation is not new. Each time a new generation of computer systems evolve, data from the old database must be regenerated and the old query set must be translated. In addition, there is a need for translation when we restructure the database. Such restructuring can be motivated by

(1) a change of the environment in which a database management system (DBMS) is used;
(2) a change in DBMS;
(3) modification of the design for efficiency reasons;
(4) a change in database semantics.

Su and Reynolds[14] studied the problem of high-level sublanguage query conversion using the relational model with SEQUEL[4] as the sublanguage, DEFINE[9] as the data description language and CONVERT[12] as the data translation language. They examined sublanguage query conversion, where query modification is due to changes in the schema and subschemas. The changes they considered were

(1) a large relation is split into several relations;
(2) several relations are combined into one;
(3) changes of the mathematical mapping between/among entities;
(4) adding or deleting entities and/or association.

Their conversion algorithm is specific to the environment they studied and serves pedagogically when an attempt is made to extend their work into the general translation problem.

The advent of lower-cost computer systems has paved the way for decentralisation of computing resources. Decentralisation has enabled designers to enhance local performance by allowing the freedom of design and software choice for each local system. Such freedom provides the necessary enhancement of local computing, but complicates the task of providing access to the database resources on a system-wide basis. In particular, it either forces the user to be aware of the location and format of all the data in the system or the computer software/hardware must have the ability to provide access to the local databases through a single data-manipulation language. Systems, such as MULTIBASE[13] provide such access, but force the user to use both the data manipulation language (DML) of MULTIBASE and the local system to get the enhanced local performance.

What is needed is a model that can provide both the dynamic query translation necessary to allow the direct communication of different DBMSs, so that the user only needs to be proficient in the DML of his/her local DBMS and the ability to provide general query translation to allow the extension of the work of Su and Reynolds.[14] In recent work on database design, a number of authors have found the hypergraph to be a useful means of modelling relational database designs.[1,2,5,6,7,11,15,18] When the hypergraph is extended to incorporate the DML operations, it is helpful in defining a general model for query translation.

In the next section, we review some basic concepts of hypergraphs and look at the use of the hypergraph in modelling the relational database. The section examines the use of the hypergraph to model both the logical design and DML operations. A translation process built on the model is described in Section 3.

2. HYPERGRAPH MODEL

We assume that the reader is familiar with the basic concepts in relational database theory, such as the basic dependencies (functional, multivalued and join) and the operators of relational algebra.[10,16] If N is the set of attributes, then we define a database scheme $R = \{R_1, R_2, \ldots, R_m\}$ to be a set of subsets of N.

A *hypergraph* is a couple $\mathscr{H} = (\mathscr{N}, \mathscr{E})$ where $\mathscr{N}$ is a set of vertices and $\mathscr{E}$ is a set of edges which are nonempty subsets of $\mathscr{N}$.[3] There is a natural correspondence between database schemes and hypergraphs.

A hypergraph $\mathscr{H} = (\mathscr{N}, \mathscr{E})$ is said to be *γ-acyclic*[6] if it contains no γ-cycle which is a sequence of the form

$$(E_1, x_1, E_2, x_2, \ldots, E_k, x_k, E_{k+1})$$

(i) $x_1, x_2, \ldots, x_k$ are distinct vertices in $\mathscr{N}$;
(ii) $E_1, E_2, \ldots, E_k$ are distinct edges in $\mathscr{E}$ and $E_{k+1} = E_1$;
(iii) $k \geqslant 3$;
(iv) x_i is in $E_i \cap E_{i+1}$, $1 \leqslant i \leqslant k$;
(v) if $1 \leqslant i \leqslant k$, then x_i is in no E_j except E_i and E_{i+1}.

The database scheme is γ-acyclic exactly when the underlying hypergraph is γ-acyclic.

Fagin[6] has defined two additional types of acyclicity for hypergraphs that are of interest: α-acyclic and β-acyclic. Graham's algorithm[10,16] repeatedly applies the following two steps on the hypergraph until neither can be applied:

(1) if x is a vertex that appears in exactly one edge E_i, then delete it from E_i;
(2) delete E_i if there is an edge E_j, $j \neq i$ such that $E_i \subseteq E_j$.

‡ To whom correspondence should be addressed.

The algorithm succeeds if it terminates with an empty set; otherwise, it fails. A hypergraph is *α-acyclic* if Graham's algorithm succeeds. A hypergraph is *β-acyclic* if every arbitrary subhypergraph is α-acyclic.

Fagin[6] has shown that γ-acyclic ⇒ β-acyclic ⇒ α-acyclic, but none of the reverse implications hold. Each type of acyclicity provides desirable properties for the corresponding database scheme. In particular, Fagin[6] has shown that any arbitrary connected set of relation schemes in a γ-acyclic database scheme defines an embedded join dependency.

Fagin *et al.*[7] use the hypergraph to model the full join dependency which defines the universal relation (UR). For example, the well-known supplier–parts database given in Fig. 2.1 is defined by the dependency set $\{\bowtie[R_1, R_2, R_3],\ S\# \rightarrow C,\ C \rightarrow S,\ S\#\ P\# \rightarrow Q\}$. The join dependency $(\text{jd})\bowtie[R_1, R_2, R_3]$ can be represented by the hypergraph of Fig. 2.2. Fagin *et al.*[7] have shown that the semantics of any database can always be represented by such a full join dependency and a set of functional dependencies (fds). Ullman[17] in his survey of universal relation assumptions denotes this as the UR/JD assumption. A second universal relation assumption we will find useful is a variant of the Pure Universal Relation Assumption.[17]

To this point, we have only represented the logical design in the hypergraph model. To be of use in translating queries, the model has to expand to incorporate the operations of a data manipulation language (DML).

In the following, we incorporate the DML commands of relational algebra into the model to fit the needs of the translation process given in the next section. The inclusion of the three fundamental operators of relational algebra – join, project and select – in the hypergraph model is described by the following set of actions.

Join. The natural join is introduced into the model by the creation of a new edge containing the combined attribute set of the two joined relations. The new edge is labelled as a join edge. To consider the more general question of the theta join requires an extension of the edge labelling process to describe the restrictions on theta.

Project. Projection is introduced into the hypergraph by inclusion of a new edge containing the projected attributes. The new edge is labelled as a projection edge.

Select. A selection of operation on a relation creates a new relation of tuples which have been defined by the given condition. The condition (F) involves a boolean combination of simple conditions. A simple condition on the hypergraph is an expression C involving: (i) operands that are vertices and constants; (ii) comparison operators; (iii) logical operators AND(∧), OR(∨) and NOT(¬). The selection formula F is in Conjunctive Normal Form (CNF), i.e. $F = C_1 \wedge C_2 \wedge \ldots \wedge C_k$. Simple conditions can be indicated in the hypergraph by labelling the appropriate attribute vertices with the simple condition. Simple conditions using the OR operator require the insertion of an OR edge labelled with the simple condition.

$$R_1(S\#, P\#, Q)$$
$$R_2(S\#, C)$$
$$R_3(C, S)$$

where $S\#$ = supplier number, $P\#$ = part number, Q = quantity, C = city located in, S = status of city.

Figure 2.1. Supplier–parts database.

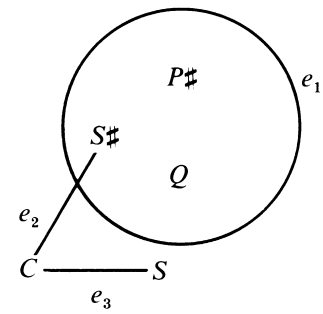


Figure 2.2. Hypergraph representation of the supplier–parts database.

Query: Select e_1 where $S\# = S1$ or $P\# = P1$ giving T_0; join T_0 and e_2 giving T_1; project T_1 over C.

Resulting hypergraph:

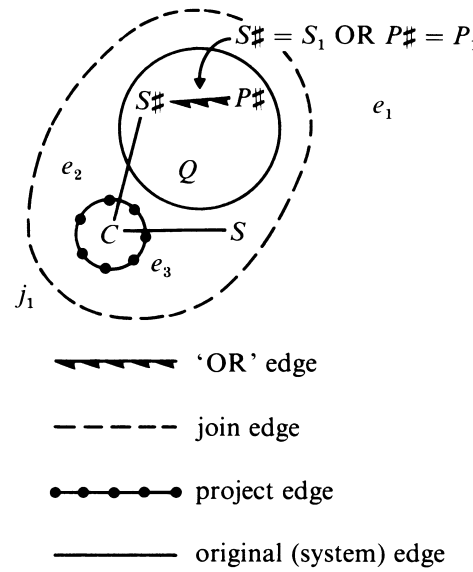


Figure 2.3. Example of including relational algebra in the hypergraph model.

Fig. 2.3 illustrates the inclusion of a relational algebra query into the hypergraph model of Fig. 2.2. The model is used in Section 3 in the query translation process.

3. TRANSLATION PROCESS

We assume that we have two relational designs over essentially the same attribute set. The designs are not the same, but attempt to support basically the same set of semantics. The two designs are represented in the hypergraph format of Section 2. The hypergraph representing the design for which the query was written is called the source hypergraph and the second is given as the target hypergraph.

To translate the query, we need three operations, as follows.

(1) Map the source query into the hypergraph space of the source hypergraph.

(2) Translate the resulting source query hypergraph into the hypergraph space of the target hypergraph.

(3) Map the target query hypergraph to the target data-manipulation language.

In the first operation, we have the task of creating a query hypergraph that has sufficient information content to provide a basis for the translation. The source query can be generalised (without loss of generality) to

$$Q = \Pi_L(\sigma_F(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)),$$

where L is the set of projected vertices, F is the selection formula in the query and is in CNF, and $R_1, R_2, \ldots, R_k$ are the relation schemes from the source design.

Two types of information are essential to the translation process. The set of attributes that represent the result of the source query (L) and any attributes that have been used in the selection formula (F) must be included in any source query hypergraph. The join sequence is not required, since the universal relation assumption has been adopted. To provide the necessary information, we have adopted a two-edge hypergraph. One edge is a projection edge (L) and the other edge contains the selection attributes labelled with the appropriate condition from F. The projection edge has been adopted to avoid marking the projection attributes. The two edges, in general, will be disjoint, although this is not required. An example of a source query and the resulting source query hypergraph is given in Fig. 3.1.

In addition to the source query hypergraph (Q_S), the translation process requires information on the target design. In particular, it requires information on the possible join sequences in the target design. We could obtain the join sequence directly from the target hypergraph ($\mathcal{H}_T$), but we have chosen the complete intersection graph (I_R)[10] for $\mathcal{H}_T$ since it is easier to operate on. To avoid confusion, we will use the term *node* when discussing sets of attributes for I_R and the term *vertex* for single attributes in the hypergraph. Algorithm 3.1 combines the source query hypergraph (Q_S) with the complete intersection graph for $\mathcal{H}_T$ to produce the join sequence for the target database. The process is similar to the creation of a join tree described by Maier,[10] but the reader will observe two important differences. First, the join sequence we require, in general, will use only a subset of the nodes in I_R. Secondly, as the algorithm is stated we cannot guarantee that the join sequence provides a lossless join.[10] As Beeri *et al.* show,[2] the existence of a join tree is equivalent to finding the original hypergraph acyclic (α-acyclic using Fagin's notation).[6] As such the join sequence is fixed for the join tree. To ensure the lossless join for the sequence that we generate, we will take the opposite approach and impose the restriction that $\mathcal{H}_T$ be γ-acyclic.[6] The γ-acyclicity restriction means that any connected subhypergraph of the $\mathcal{H}_T$ defines an embedded join dependency.[6] The restriction is imposed here in order to simplify our discussion. The restriction can be removed by expanding the join sequence set produced by Algorithm 3.1 either to a hinge[8] or to a maximal object.[11]

To create the trees in Algorithm 3.1, we use a variation of the breath-first search (BFS) to determine the join sequence for $\mathcal{H}_T$. The algorithm supplements BFS by including a label for each path in the search tree and a set, which we will call the adjustment set (A). The algorithm creates a search tree (we will call the tree an adjusted BFS tree [ABFS]) for each node in I_R which contains an attribute of $\mathcal{S}$ (the set of source query attributes). The search tree path labels are used to prune or delay the expansion of subtrees where the unused nodes that are adjacent to the current endpoint of the path do not contribute any new query attributes ($\mathcal{S}$) to the path label. Any nodes falling into this class are stored in the adjustment set (A) with a pointer to the position

Query: join $(ABDF) \bowtie (BCEG)$ giving T_0; select T_0 where $C = 1$ AND $D = 2$ AND $E = \alpha$ giving T_1; project T_1 over A, B, C giving RESULT.

Hypergraph (Q_S):

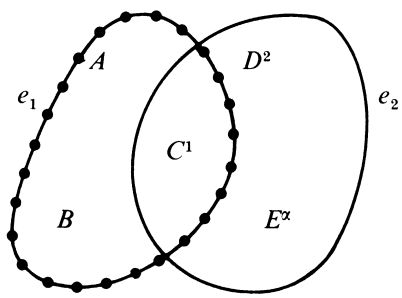


Figure 3.1. Example of a source query hypergraph.

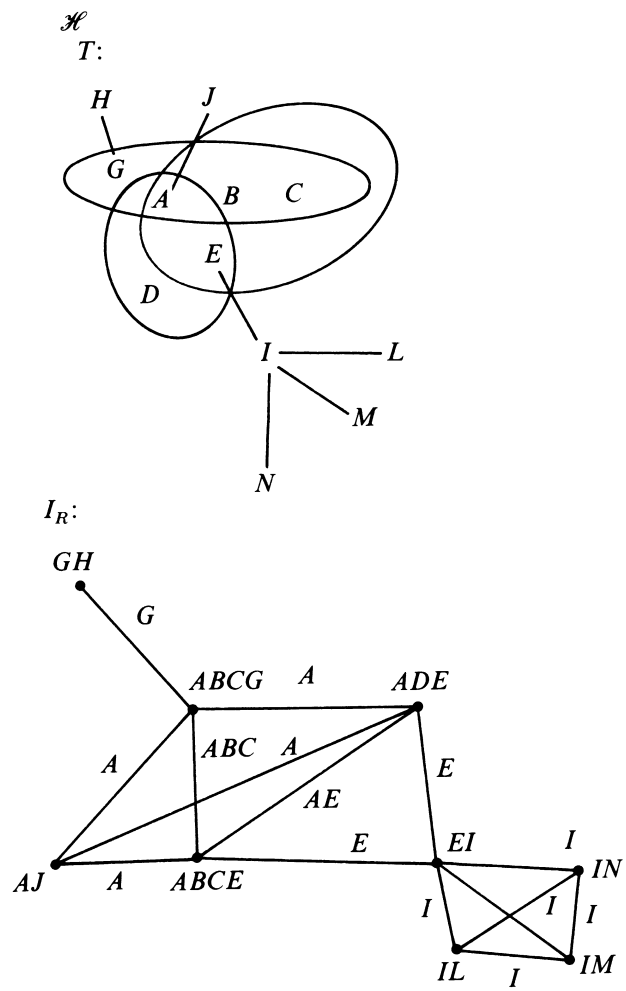


Figure 3.2. Sample hypergraph and its complete intersection graph.

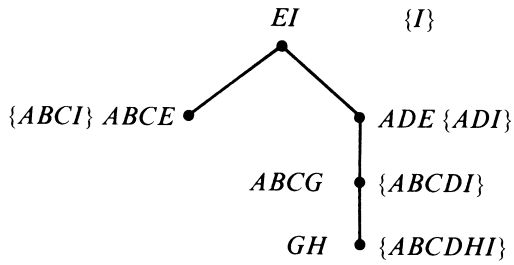
where they could be added to the search tree. The expansion continues until

$$\mathcal{S} = \bigcup_{i=1}^{m} P_i,$$

where P_i represents a path label for each of the m paths. In the event that we can no longer expand the search tree and $\mathcal{S} \neq \bigcup_{i=1}^{m} P_i$, then we take nodes from A and restart the process. Examples 3.1 and 3.2 illustrate the use of the process for the complete intersection graph of Fig. 3.2.

Example 3.1

Create the search tree for EI of the I_R of Fig. 3.2, where $\mathcal{S} = \{A, B, C, D, H, I\}$.



Adjustment set (A)
(parent in parentheses)

$\{IL(EI), IM(EI), IN(EI), AJ(ABCE)\}$

Notice that to expand $ABCE$ with either $ABCG$ or AJ is of no value (i.e. adds no new $\mathcal{S}$ attribute to path label) in this tree, but by saving $ABCG$ in the adjustment set we are able to use it in the expansion of ADE. ■

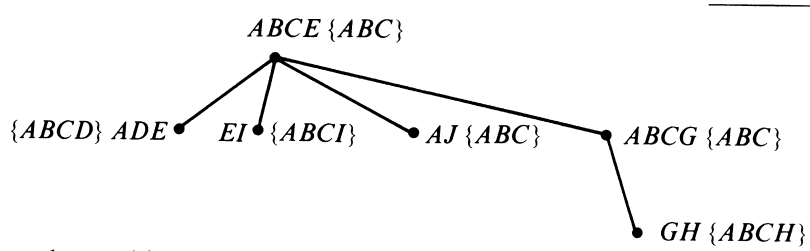
Example 3.2

Create the search tree for $ABCE$ of the I_R of Fig. 3.2 where $\mathcal{S} = \{A, B, C, D, H, I\}$.

Adjustment set (A)

$\{AJ(ABCE), ABCG(ABCE), IL(EI), IM(EI), IN(EI)\}$

At this point the process stalls, since we have no node in the adjustment set that can add any $\mathcal{S}$ attribute, and GH cannot be reached. Choose an element from A and continue the process. We have



as the resulting search tree. ■

From the examples, it is clear that the adjustment set plays two roles in the expansion of the ABFS tree. First, it is used to retain nodes where the desired expansion is simply a matter of the order in which the nodes are examined (Example 3.1). Second, it is used to restart the expansion of the ABFS tree when our pruning does not allow creation of a tree which includes all the query attributes ($\mathcal{S}$). Fig. 3.3 illustrates that this process may be required to generate the optimal ABFS tree. By an optimal ABFS tree, we mean one which has the minimum weight (minimum join sequence) over all possible ABFS trees for a given root. Note that this approach does not guarantee that we will generate the optimal ABFS tree for a given root (Fig. 3.4). The order of search in the procedure create-ABFS-tree for Algorithm 3.1 is not necessarily optimal. The order in which the nodes are considered determines whether or not we can generate an optimal ABFS tree for a given root. In Fig. 3.4 the algorithm failed to create an optimal ABFS tree for the root EI. However, if the node ADE is considered prior to ABE, we would get an optimal ABFS tree. In the tests conducted on Algorithm 3.1, we have not generated a hypergraph that created a non-optimal ABFS tree for each node in the optimal join sequence. It seems unlikely that such an event would occur in the practical environment. Such a possibility does not affect the complexity of the translation process, but can create a non-optimal join. The creation of a non-optimal ABFS tree does not cause serious problems, since a redundant join can be removed at a later stage.

Once the ABFS tree is created, we need to be able to determine the join sequence defined by the tree. Our approach is to take the root and a set of paths connected to the root such that the union of the path labels includes the attributes found in $\mathcal{S}$. Based on this approach, the complete set of ABFS trees and their assigned weights for the I_R of Fig. 3.2 are given in Fig. 3.5. Note that nodes such as AJ, that are only adjacent to nodes containing a superset of the $\mathcal{S}$ attributes found in AJ, are assigned the weight MAX to remove them from consideration. The join sequence defined by one of the ABFS trees having the minimum weight is taken as the appropriate join sequence for the target design and passed on to Algorithm 3.2.

Algorithm 3.1 uses a three-tier approach to determine the appropriate paths to select in each ABFS tree. The first level examines the path labels for $\mathcal{S}$ attributes that only occur in a single label. The paths are marked as part of the join sequence. For most of the test hypergraphs that we examined this was sufficient to generate the appropriate join sequence. However, as can be seen from Fig. 3.6, it is not adequate to generate all join sequences.

The second level of the path-selection algorithm operates on nodes that are added at the terminal level

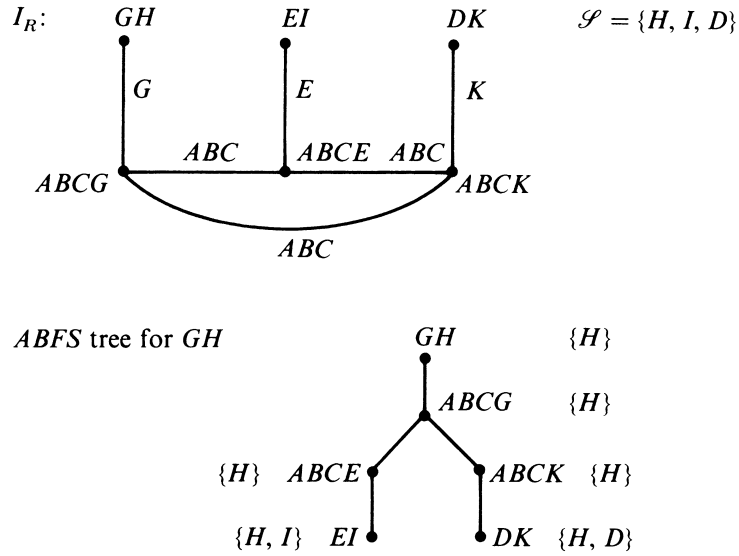


Figure 3.3. Example showing that elements from the adjustment set must be used in the expansion of the ABFS tree.

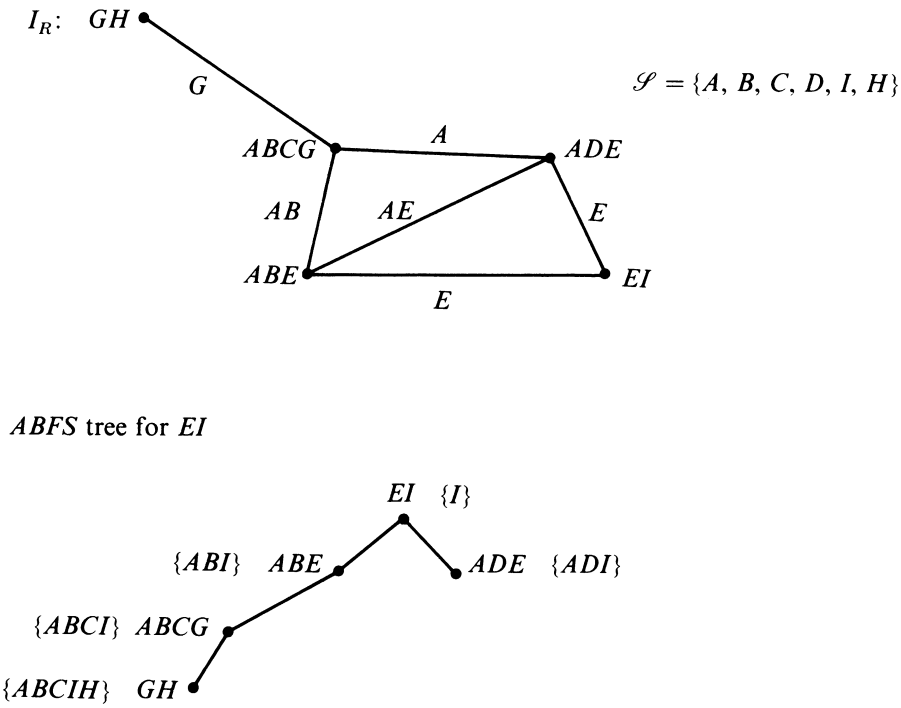


Figure 3.4. Example illustrating the creation of a non-optimal ABFS tree.

and have the same parent. The final step forces the selection of any remaining query attributes that have not been added to the join sequence by the first two steps. (In our test data, the first two steps were sufficient in all cases to select the paths necessary to perform the required join when $\mathcal{H}$ is γ-acyclic.) The step is clearly required when the γ-acyclic restriction is dropped.

The algorithm has performed well on test data. While it is true that for any particular ABFS tree we can choose a non-optimal path (as well as a non-optimal ABFS tree), it has not kept us, in our test data, from coming up with a good join sequence. The low possibility of generating non-optimal join sequences for all of the relations in the desired join sequence leads us to believe the algorithm will perform well in practice. Since the ABFS trees are generally related in size to the number of attributes in the query, the generation of the trees is expected to be much less than the worst case. In the worst case we need to generate an ABFS tree containing all of the nodes in I_R (when query nodes fall on the extreme end points of the hypergraph), but in general one expects queries to be much more localised within the hypergraph.

The constraints placed on the target hypergraph are not sufficient to guarantee uniqueness, as we can see from the hypergraph in Fig. 3.7. For $\mathcal{H}$, either $\{HL, LA, ABCE, AK, KI\}$ or $\{HL, LA, ABCG, AK, KI\}$ will produce the desired join sequence when $\mathcal{S} = \{A, B, C, H, I\}$.

The join sequence is used in Algorithm 3.2 to produce the target query hypergraph (Q_T). The target query hypergraph (Q_T) can then be used to create the target

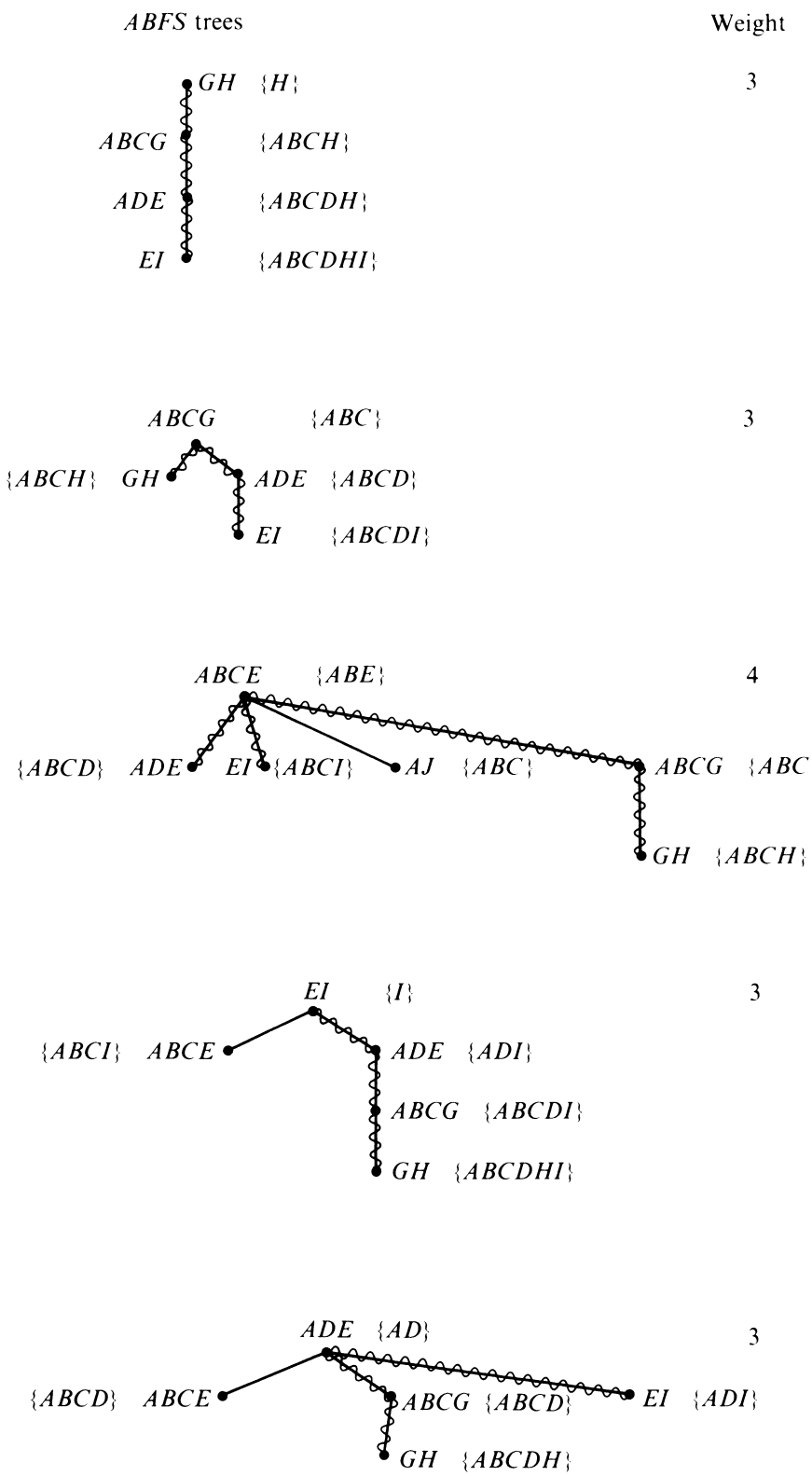
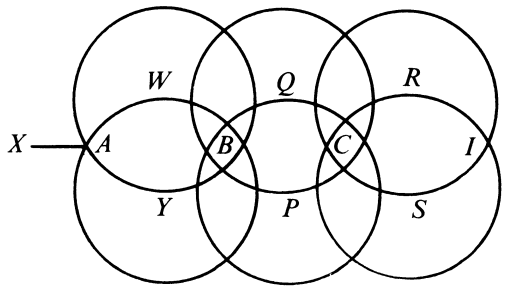


Figure 3.5. ABFS trees and weights produced by Algorithm 3.1 for the target hypergraph of Fig. 3.2.

query. In order to create the target query, a mapping algorithm is required to map Q_T into the appropriate DML. Algorithm 3.3 provides a mapping for relational algebra. Similar algorithms can easily be given for any required DML. Algorithm 3.1 is the dominant algorithm of the three. In the worst case, the attributes in $\mathscr{S}$ could be involved in all edges of $\mathscr{H}_T$ and the algorithm would need to generate one ABFS tree for each node in the complete intersection graph for $\mathscr{H}_T$. Each ABFS tree could require that n nodes be included, and in the worst case $n-1$ nodes must be examined to add a new node to the ABFS tree. Thus the algorithm requires time proportional to n^3 in the worst case, where n is the number of nodes in the complete intersection graph for the target

$\mathscr{H}_T$:



$\mathscr{S} = \{A, B, C, I\}$

ABFS tree for AX

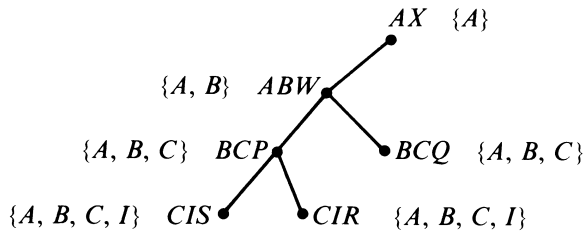
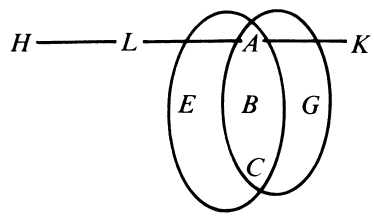


Figure 3.6. Example to show we must break ties.

$\mathscr{H}_T$:



$\mathscr{S} = \{A, B, C, H, I\}$

I_R:

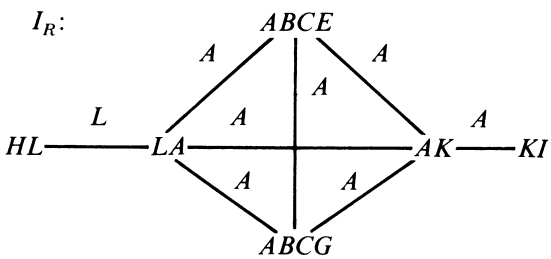


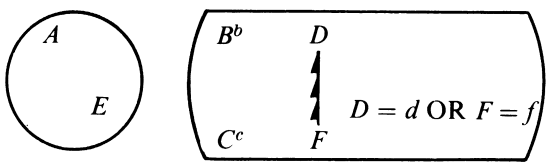
Figure 3.7. Example of non-unique optimal join sequence, namely, {*HL, LA, ABCE, AK, KI*} or {*HL, LA, ABCG, AK, KI*}.

hypergraph, $\mathscr{H}_T$. An illustration of the complete process is given in Example 3.3.

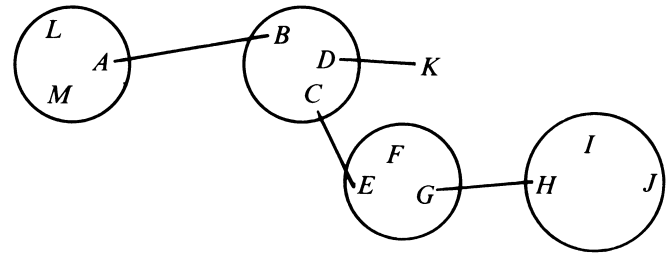
Example 3.3

Translation example.

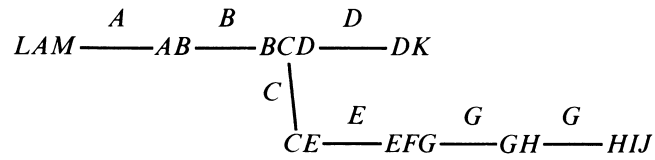
* Source Query Hypergraph (Q_S)



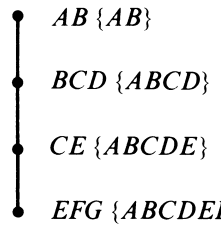
* Target Hypergraph ($\mathscr{H}_T$)



* I_R for $\mathscr{H}_T$



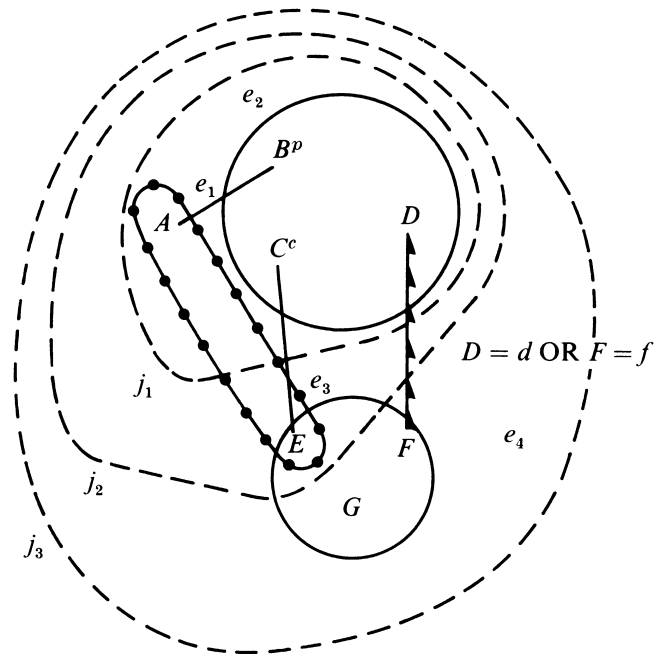
* Join sequence



$\mathscr{S} = \{A, B, C, D, E, F\}$

$\rightarrow \{AB, BCD, CE, EFG\}$ is the join sequence generated by Algorithm 3.1.

* Target Query Hypergraph (Q_T) generated by Algorithm 3.2.



* Query generated by Algorithm 3.3

$$\Pi_{A,E}(\sigma_{B=b \text{ AND } C=c \text{ AND } (D=d \text{ OR } F=f)})(e_1 \bowtie e_2 \bowtie e_3 \bowtie e_4)$$

It remains to examine the equivalence of the source and the target query produced by the process. We will take the view that two queries are *equivalent*, if they

produce the same result given that the two universal relations (source and target) produce the same result when projected over (the query attributes). In the following, it is assumed that the system relations are projections of the universal relations at the time of comparison.[17] In addition, we make the assumption that the source query does not use a disjoint join sequence. (If we take the join $R_1 \bowtie R_2$ and $R_1 \cap R_2 = \varnothing$, then the join is equivalent to the cartesian product.) Such an assumption is reasonable, since a disjoint join will produce a lossy join and is likely to produce a result with incorrect data relationships. However, it should be noted that we are giving up something with this assumption. A disjoint join can be used to define a union operation when the OR is allowed to span over more than one relation.

The following remark examines the question of equivalence using relational algebra. Relational algebra is used for its simplicity here, with the notation that any query language that is relationally complete[16] can be represented using relational algebra.

Proposition. Let $\mathcal{H}_S$ and $\mathcal{H}_T$ be γ-acyclic hypergraphs defining the source database and target database, respectively. The target query produced by Algorithms 3.1, 3.2 and 3.3 is equivalent to the source query.

Proof. Since the two hypergraphs, $\mathcal{H}_S$ and $\mathcal{H}_T$, are γ-acyclic, any connected subhypergraph forms an embedded join dependency.[6] Therefore the join sequences used in the source and target queries produce a lossless join. Let r_Q represent the result of the join sequence required by the source query and s_Q represent the result of the join sequence for the target query.

Due to the lossless join property, the data relationships between the query attributes ($\mathcal{S}$) will be the same in r_Q and s_Q as in their respective universal relations. As long as the two universal relations support the same data relationships for the query attributes, we have

$$\pi_{\mathcal{S}}(r_Q) = \pi_{\mathcal{S}}(s_Q).$$

The selection operator applies a selection condition in CNF and will achieve the same result whether it is applied as a sequence of k selections with condition c_i to relations used in the join process or as one selection applied to the final join result with the c_i ANDed together. Therefore, we have

$$\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_k}(\pi_{\mathcal{S}}(r_Q)) = \sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_k}(\pi_{\mathcal{S}}(s_Q)),$$

since all of the conditions c_i, $i = 1, \ldots, k$, are applied to the query attributes ($\mathcal{S}$).

As the two relations are projected over the same set of query vertices to achieve the result for the query, we have both the source query and the target query producing the same result, given the same data relationships for the query attributes ($\mathcal{S}$) in their respective universal relations. Therefore, the two queries are equivalent. ■

We note that the two universal relations do not have to support the same set of attributes. They only need to support the attributes in $\mathcal{S}$ and have the same data relationships for the attributes in $\mathcal{S}$.

4. CONCLUSION

A generalised query translation process for query translation between two relational database schemes has been presented. The process uses the hypergraph model of the target database scheme to provide sufficient information to generate the appropriate join sequence.

The process is sufficiently general to provide the basis of generalising the work of Su and Reynolds,[14] as well as providing a means of dynamic query translation. The work in Section 3 was based on the conversion to relational algebra. The process can easily be extended to other data manipulation languages by creating variations of Algorithm 3.3 for the appropriate DML used with the relational model.

Our current work centres on the examination of the optimality considerations of choosing the join sequence in Algorithm 3.1. The algorithm has performed well on test designs, but more work needs to be done in this area.

Acknowledgements

We would like to thank the reviewer for his many suggestions. His comments have resulted in several improvements over our original manuscript.

REFERENCES

1. C. Beeri, R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman and M. Yannakakis, Properties of acyclic database schemes. *Proceedings of the 13th ACM Symposium on the Theory of Computing, Milwaukee*, 352–362 (1981).
2. C. Beeri, R. Fagin, D. Maier and M. Yannakakis, On the desirability of acyclic database schemes. *Journal of ACM* **30**, 479–513 (1983).
3. C. Berge, *Graphs and Hypergraphs*. North-Holland, Amsterdam (1973).
4. D. Chamberlin and R. Boyce, SEQUEL: a structured English query language. *Proceedings of ACM SIGMOD Workshop on Data Descriptions, Access, and Control, Ann Arbor, Michigan*, 249–264 (1974).
5. K. Chase, Join graphs and acyclic database schemes. *Proceedings of the 6th International Conference on Very Large Databases, Montreal*, 95–100 (1980).
6. R. Fagin, Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of ACM* **30**, 514–550 (1983).
7. R. Fagin, A. O. Mendelzon and J. D. Ullman, A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems* **7**, 343–360 (1982).
8. M. Gyssens and J. Paredaens, A decomposition methodology for cyclic databases. In *Advances in Database Theory*, edited H. Gallaire, J. Minker and J. M. Nicolas, vol. 2, pp. 85–122 (1984).
9. B. C. Housel, D. Smith, N. C. Shu and V. Y. Lum, DEFINE: a nonprocedural data description language for defining information easily. *Proceedings of the ACM Pacific Regional Conference, San Francisco*, 62–70 (1975).
10. D. Maier, *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland (1983).
11. D. Maier and J. D. Ullman, Maximal objects and the semantics of universal relation databases. *ACM Transactions on Database Systems* **8**, 1–14 (1983).
12. N. C. Shu, B. C. Housel and V. Y. Lum, CONVERT: a high level translation definition language for data conversion. *Communications of ACM* **18**, 557–567 (1975).

13. J. M. Smith, P. Berstein, U. Dayal, N. Goodman, T. Landers, K. Lin and E. Wong, MULTIBASE: integrating heterogeneous distributed database systems. *Proceedings of AFIPS National Computer Conference* **50**, 487–499 (1981).
14. S. Y. W. Su and M. J. Reynolds, Conversion of high level sublanguage queries to account for database changes. *Proceedings of the 2nd International Conference on Very Large Databases, Brussels,* 143–157 (1976).
15. R. E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing* **13**, 566–579 (1981); Addendum: **14**, 254–255 (1985).
16. J. D. Ullman, *Principles of Database Systems*, 2nd edn. Computer Science Press, Rockville, Maryland (1982).
17. J. D. Ullman, The universal relation strikes back. *Proceedings of the ACM Symposium on the Principles of Database Systems, Los Angeles*, 10–22 (1982).
18. M. Yannakakis, Algorithms for acyclic database schemes. *Proceedings of the 7th International Conference on Very Large Databases, Cannes, France*, 82–94 (1981).

Algorithm 3.1. Creation of join sequence

```
begin
  S := the set of attributes in the source query;
  X := the set of nodes of I_R containing elements of S;
  joincount := n; {where n is the size of I_R}
  while X ≠ ∅ do
    begin
      choose x_i ∈ X;
      create-ABFS-tree (x_i);
      choose-path (x_i);
      if path-length < joincount then
      begin root := x_i;
            save ABFS tree;
            joincount := path-length
      end;
      X := X − {x_i}
    end
end.

procedure create-ABFS-tree (root);
  begin
    Y := {root}; Initialise ABFS tree; A := ∅;
    path (root) := S attr (root);
    while S ≠ ∪ (i ∈ ABFS) path(i) and there are nodes in I_R
          that have not been traversed yet do
    begin
    while Y ≠ ∅ and S ≠ ∪ (i ∈ ABFS) path(i) do
    begin
      choose y ∈ Y; {first y in Y}
      let J := nodes adjacent to y and not in ABFS
           tree;
    while J ≠ ∅ and S ≠ ∪ (i ∈ ABFS) path(i) do
    begin
      choose j ∈ J;
    if S attr(j) ⊈ path(y)
    then
    begin {expand ABFS tree}
      Y := Y ∪ {j}; {append to end of Y}
      mark j is in ABFS;
    if j ∈ A then remove j from A;
      path(j) := path(y) ∪ S attr(j);
      pointer(j) := y
    end
    else
    begin
    if j is not in set(A) then
    begin
      place j and a pointer to y in the adjustment
           set(A);
      mark j is in A
    end
    end;
      J := J − {j}
    end;
      Y := Y − {y}
    end;
    if S ≠ ∪ (i ∈ ABFS) path(i) then
    begin
      choose a ∈ A;
      A := A − {a};
      place a in ABFS tree using pointer from A;
      mark a is in ABFS tree;
      Y := {a}
    end
    end
end;

procedure choose-path(x_i);
  begin
    count the number of occurrences for each B ∈ S in the
         path labels of the terminal nodes; {Freq(B)} Let
         L := the set of path labels for the terminal nodes;
         L' := ∅;
    while L ≠ ∅ and S ≠ ∅ do
    begin {Examine the path labels for S attributes that
         occur in a single path label in L}
      choose l ∈ L; L := L − {l};
if ∃ B ∈ path(l) such that Freq(B) = 1
then
begin
  mark the path whose path label is 1;
  reduce S and the path labels in L by the attributes
       in l;
  remove any empty path labels from L created by
       the reduction
end
else
  L' := L' ∪ {l}
end;
  L'' := ∅;
while L' ≠ ∅ and S ≠ ∅ do
begin {operate on nodes that are added at the terminal
       level with the same parent}
  choose l ∈ L';
  let T := the S attributes added by the terminal
       node of the path with label l;
while T ≠ ∅ do
begin choose t ∈ T; T := T − {t};
if the contributing nodes for t have the same parent
then
begin
  choose contributing path adding the most new S
       attributes, in case of tie choose either;
```

mark chosen path;
reduce T, $\mathcal{S}$, and the labels in L′ by the attributes chosen path label;
remove any empty labels from L′ created by the reduction
end
end;
if $l \neq \varnothing$ **then**
begin
$L'' := L'' \cup \{l\}$;
$L' := L' - \{l\}$
end
end;
while $\mathcal{S} \neq \varnothing$ **do**
begin {*select any remaining query attributes that have not been added to the join sequence*}
choose $l \in L''$;
choose lowest frequency attribute added at the terminal step;
choose path contributing the most new $\mathcal{S}$ attributes, in case of tie use the shortest path;
mark the chosen path;
reduce $\mathcal{S}$ and the labels in L″ by the attributes in the chosen path label;
remove any empty labels from L″ created by the reduction
end;
path-length := number of edges in marked paths
end;

Algorithm 3.2. Creation of the target query hypergraph (Q_T)

begin
$R := \varnothing$; $R' := \varnothing$;
let $\mathcal{S}$:= *vertices in the source query hypergraph*;
let J := *nodes in the join sequence*;
assign the first element of J to R;
let $J := J - R$; *weight* := 1;
while $\mathcal{S} \not\subseteq R$ **do**
begin
assign the first element of J to R′; {*note that a node in I_R represents an edge in Q_T*}
generate a join edge for the attributes in $R \cup R'$ *labelled with* j_{weight};
weight := *weight* + 1;
let $R := R \cup R'$; $J := J - R'$
end;
copy labels from the vertices in Q_S to the vertices in R;
insert the "OR" edges in R from Q_S;
insert the projection edge in R from Q_S
end.

Algorithm 3.3. Mapping target query hypergraph to relational algebra

begin
$i := 1$; {*join edge weight*}
$F := \varnothing$; {*selection condition*}
$J := \varnothing$; {*set of joined relation schemes*}
P := *the set of vertices in the project edge*;
if *join edges exist*
then
begin
choose join edge j_i; {*use first join edge* j_1}
let J := *the join of the system edges in* j_i; {*perform joins*}
if *any vertices in* j_i *are labelled* {*create selection formula for* j_1}
then
let F := *the AND of all labels within* j_i;
$i := i + 1$ {*point to next join edge*}
end;
while *a join edge* j_i *exists* **do**
begin
let J := *the join of the system edges in* j_i; {*perform joins*}
if *any vertices in* $j_i - j_{i-1}$ *are labelled* {*expand selection formula*}
then
begin
let condition be the AND of all labels in $j_i - j_{i-1}$;
$F := F$ *AND condition*
end;
$i := i + 1$ {*point to next join edge*}
end;
generate '$\Pi_P(\sigma_F(J))$'
end.

Announcement

1–5 AUGUST 1988

10th Congress of the International Ergonomics Association, Sydney, Australia

The 10th International Ergonomics Congress, to be held in Sydney from 1 to 5 August 1988, has released its provisional programme and registration details. An impressive 32-page booklet, it contains details on registration, the scientific programme, keynote speakers, social and accompanying persons' programmes, and associated meetings.

'Designing a Better World' is the challenging congress theme, and an imaginative programme reflects this challenge.

Two post-Congress tours are offered: one to Australia's 'Red Centre' and famous Kakadu National Park in 'Crocodile Dundee' territory, the other to the beautiful Great Barrier Reef in Northern Queensland. A weekend escape on the waterways close to Sydney is offered for those with limited time to take a break.

The programme also contains general information about Australia, travel and accommodation details.

Enquiries to:
IEA88 Secretariat, PO Box 380, Spit Junction, NSW 2088, Australia. Tel. 61 2 969 1400. Fax 61 2 908 4982.