Abstract. An increasing number of applications depend on efficient storage and analysis features for XML data. Hence, query optimization and efficient evaluation techniques for the emerging XQuery standard become more and more important. Many XQuery queries require nested expressions. unnesting them often introduces binary grouping. We introduce several algorithms implementing binary grouping and analyze their time and space complexity. Experiments demonstrate their performance.

1 Motivation

Optimization and efficient evaluation of queries over XML data becomes more and more important because an increasing number of applications work with XML data. In XQuery – the emerging standard query language for XML – queries including restructuring or aggregation often require nested queries. For example, the following query returns for each of the fifty richest persons of the world the number of countries with smaller gross domestic product (GDP) than the person’s total capital.

```xml
for $p in document("richest-fifty.xml")//person
return
  <result>
    <person> { $p/name } </person>
    <count-richer> {
      count(for $c in document("countries.xml")//country
       where $p/capital gt $c/gdp
       return $c)
    }
  </count-richer>
</result>
```

This query combines data of two different documents and performs grouping and aggregation over the XML data. Note that each country can contribute to the count of multiple persons, and that a non-equality predicate is used to relate items from both documents.
Direct nested evaluation of this query is highly inefficient because for each person the nested FLWR expression is evaluated, demanding a scan of the countries document. Fortunately, the query can be unnested introducing binary grouping [17]. Moreover, optimizers can then apply algebraic equivalences to further improve performance. However, efficient implementations for binary grouping are not available yet. If they were, the optimizer could choose among them, ensuring an efficient query evaluation. We fill this gap and present several main-memory algorithms for implementing binary grouping. Further, we analyze their time and space complexity. The different algorithms will require different conditions to hold. Enumerating them then enables the query optimizer to select the most efficient implementation of binary grouping for a given situation. Experiments demonstrate that performance can be improved by orders of magnitude. Due to space constraints, we restrict ourselves to the formulation of algorithms working on sets of tuples. However, an extension to bags or sequences is not difficult (see [18]). Let us stress that binary grouping is useful not only in the context of XQuery. It has also been successfully applied to unnest nested OQL-queries [4, 20] and to evaluate complex OLAP queries [2].

The paper is structured as follows. Section 2 presents the definition of binary grouping and surveys properties of predicates and aggregate functions. They form the basis for the selection of an efficient implementation for the binary grouping operator. The main contribution of this paper – Section 3 – introduces several algorithms for binary grouping and analyzes their time and space complexity. Exemplary performance results are given in Section 4. More detailed experimental data is presented in [18]. Before concluding this paper, Section 5 reviews related work.

2 Preliminaries

2.1 The Algebra

We will only present the operators needed for our exposition. For an extensive treatment of our algebra we refer to [4]. Our framework is extendible to sequences as required in XQuery (cf. [17] for this algebra and related work).

The algebra works on sets of unordered tuples. Each tuple contains a set of variable bindings representing the attributes of the tuple. Single tuples are constructed by using the standard $\lbrack \cdot \rbrack$ brackets. The concatenation of tuples and functions is denoted by $\circ$. The set of attributes defined for an expression $e$ is defined as $A(e)$. The set of free variables of an expression $e$ is defined as $F(e)$.

For an expression $e_1$ possibly containing free variables and a tuple $t$, $e_1(t)$ denotes the result of evaluating $e_1$ where bindings of free variables are taken from variable bindings provided by $t$ – this requires $F(e_1) \subseteq A(t)$. Note that this can also be used for function application. We denote NULL values by $\perp$.

The semantics of the binary grouping operator is defined by the map operator ($\chi$) and the selection ($\sigma$). If their input is the empty set ($\emptyset$), their output is also empty.

Let us briefly recall selection with predicate $p$ defined as $\sigma_p(e) := \{x | x \in e, p(x)\}$ and map defined as $\chi_{a:e_2}(e_1) := \{y \circ [a : e_2(y)] | y \in e_1\}$. The latter