

# 1 Spreadsheet Data Extractor (SDE): A Performance-Optimized, 2 User-Centric Tool for Transforming Semi-Structured Excel 3 Spreadsheets into Relational Data 4

5 ANONYMOUS AUTHOR(S)  
6

7 SUBMISSION ID:  
8

9 Spreadsheets are ubiquitous tools used across various domains. Despite their widespread use, analyzing and  
10 utilizing data stored in spreadsheets poses significant challenges due to their semi-structured nature. Data  
11 in spreadsheets are often formatted primarily for human readability, employing layouts and styles that are  
12 easily understood by people but are difficult for automated systems to interpret. While these unstructured  
13 formats offer advantages—such as providing an easily comprehensible hierarchy of metadata—they complicate  
14 automated data extraction. This paper introduces the Spreadsheet Data Extractor (SDE), an open-source tool  
15 designed to convert semi-structured spreadsheet data into structured formats without requiring programming  
16 knowledge. Building upon previous work, we have enhanced the SDE with incremental loading of worksheets,  
17 accurate rendering of cell dimensions by directly parsing the Excel file's XML content, and performance  
18 optimizations to handle large datasets efficiently. We compare our tool with existing solutions and demonstrate  
19 its effectiveness through performance evaluations, highlighting its potential to facilitate efficient and reliable  
data extraction from diverse spreadsheet formats.

20 CCS Concepts: • Applied computing → Spreadsheets; • Information systems → Data cleaning;  
21 Software and its engineering → Extensible Markup Language (XML); • Human-centered computing  
22 → Graphical user interfaces; • Theory of computation → Data compression.

23 Additional Key Words and Phrases: Spreadsheets, Data cleaning, Relational Data, Excel, XML, Graphical user  
24 interfaces  
25

## 26 ACM Reference Format:

27 Anonymous Author(s). 2025. Spreadsheet Data Extractor (SDE): A Performance-Optimized, User-Centric Tool  
28 for Transforming Semi-Structured Excel Spreadsheets into Relational Data. In *Proceedings of 2025 International  
29 Conference on Management of Data (ACM PODS '25)*. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/XXXXXXX.XXXXXXX>

## 31 32 1 INTRODUCTION

33 Spreadsheets are ubiquitous tools used across various domains, including healthcare [4], nonprofit  
34 organizations [11, 14], finance, commerce, academia, and government [7]. Despite their widespread  
35 use, analyzing and utilizing data stored in spreadsheets poses significant challenges for data  
36 analysis and utilization, primarily due to their semi-structured nature. Data within spreadsheets  
37 are frequently organized for human readability, featuring layouts with empty cells, merged cells,  
38 hierarchical headers, and multiple tables. While these formats enhance comprehensibility for  
39 users, they impede machine readability and automated data processing. This semi-structured

40  
41 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee  
42 provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the  
43 full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored.  
44 Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires  
45 prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

46 ACM PODS '25, June 22–27, 2025, Berlin, Germany

47 © 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

48 ACM ISBN ...\$15.00

49 <https://doi.org/10.1145/XXXXXXX.XXXXXXX>

organization complicates the extraction of meaningful insights, especially when attempting to integrate spreadsheet data into more robust and scalable data management systems.

The reliance on spreadsheets as ad-hoc solutions poses several limitations:

- **Data Integrity and Consistency:** Spreadsheets are prone to errors, such as duplicate entries, inconsistent data formats, and inadvertent modifications, which can compromise data integrity.
- **Scalability Issues:** As datasets grow in size and complexity, spreadsheets become less efficient for data storage and retrieval, leading to performance bottlenecks.
- **Limited Query Capabilities:** Unlike databases, spreadsheets lack advanced querying and indexing features, restricting users from performing complex data analyses.
- **Integration Challenges:** Integrating spreadsheet data with other systems, applications, or databases is often cumbersome and error-prone, limiting interoperability.

Transitioning from these ad-hoc spreadsheet solutions to standardized database systems offers numerous benefits:

- **Enhanced Data Integrity:** Databases enforce data validation rules and constraints, ensuring higher data quality and consistency.
- **Improved Scalability:** Databases are designed to handle large volumes of data efficiently, supporting complex queries and transactions without significant performance degradation.
- **Advanced Querying and Reporting:** Databases provide powerful querying languages like SQL, enabling sophisticated data analysis and reporting capabilities.
- **Seamless Integration:** Databases facilitate easier integration with various applications and services, promoting interoperability and data sharing across platforms.

Given the abundance of existing spreadsheet data and the clear advantages of database systems, there is a pressing need for tools that can bridge the gap between these two formats. Automated and accurate data extraction from spreadsheets into relational database formats is essential for organizations to leverage their data assets effectively.

Previous work by Aue et al. introduced a tool that facilitates data extraction from Excel files [1]. While effective, their solution faced performance issues and inaccuracies in rendering cell dimensions, limiting its usability with large and complex datasets.

In this paper, we aim to transform semi-structured spreadsheet data into machine-readable formats by building upon their work. We present the Spreadsheet Data Extractor (SDE), an enhanced tool that enables users to define data hierarchies through cell selection without any programming knowledge. Our enhancements address key limitations of the existing solution, making data extraction from complex spreadsheets more efficient and user-friendly.

## 1.1 Contributions

Our main contributions are as follows:

- (1) We release the SDE under the open-source GNU General Public License v3.0, promoting community access and collaboration [2].
- (2) We implement incremental loading of worksheets to enhance performance, allowing the tool to handle large Excel files efficiently.
- (3) We accurately render row heights and column widths by parsing XML data, ensuring that the spreadsheet's visual representation closely matches that of Excel.
- (4) We optimize the rendering engine to draw only the visible cells, significantly improving performance when dealing with large datasets.
- (5) We integrate the selection hierarchy, worksheet view, and output preview into a unified interface, streamlining the user experience.

## 99    2 RELATED WORK

100    The extraction of relational data from semi-structured documents, particularly spreadsheets, has  
101    garnered significant attention due to their ubiquitous use across domains such as business, govern-  
102    ment, and scientific research. Several frameworks and tools have been developed to address  
103    the challenges of converting flexible spreadsheet formats into normalized relational forms suit-  
104    able for data analysis and integration. Notable among these are **DeExcelerator** [8], **XLIndy** [9],  
105    **FLASHRELATE** [3], **Senbazuru** [5], **TableSense** [6], and the approach by Aue et al. [1], on which  
106    our work builds.

107   

### 108    2.1 Aue et al.'s Converter

109    Aue et al. [1] developed a tool to facilitate data extraction from Excel spreadsheets by leveraging  
110    the Dart excel package [?] to process .xlsx files. This tool allows users to define data hierarchies  
111    by selecting relevant cells containing data and metadata. However, the approach faced significant  
112    performance bottlenecks due to the excel package's requirement to load the entire .xlsx file into  
113    memory, resulting in slow response times, particularly for large files

114    In addition to memory issues, the tool calculated row heights and column widths based solely on  
115    cell content, ignoring the dimensions specified in the original Excel file. This led to rendering  
116    discrepancies between the tool and the original spreadsheet. Furthermore, the tool rendered all  
117    cells, regardless of their visibility within the viewport, significantly degrading performance when  
118    handling worksheets with large numbers of cells.

119   

### 120    2.2 DeExcelerator

121    Eberius et al. [8] introduced **DeExcelerator**, a framework that transforms partially structured  
122    spreadsheets into first normal form relational tables using heuristic-based extraction phases. It  
123    addresses challenges such as table detection, metadata extraction, and layout normalization. While  
124    effective in automating normalization, its reliance on predefined heuristics limits adaptability to  
125    heterogeneous or unconventional spreadsheet formats, highlighting the need for more flexible  
126    approaches.

127   

### 128    2.3 XLIndy

129    Koci et al. [9] developed **XLIndy**, an interactive Excel add-in with a Python-based machine learning  
130    backend. Unlike DeExcelerator's fully automated heuristic approach, XLIndy integrates machine  
131    learning techniques for layout inference and table recognition, enabling a more adaptable and  
132    accurate extraction process. XLIndy's interactive interface allows users to visually inspect extraction  
133    results, adjust configurations, and compare different extraction runs, facilitating iterative fine-tuning.  
134    Additionally, users can manually revise predicted layouts and tables, saving these revisions as  
135    annotations to improve classifier performance through (re-)training. This user-centric approach  
136    enhances the tool's flexibility, allowing it to accommodate diverse spreadsheet formats and user-  
137    specific requirements more effectively than purely heuristic-based systems.

138   

### 139    2.4 FLASHRELATE

140    Barowy et al. [3] presented **FLASHRELATE**, an approach that empowers users to extract struc-  
141    tured relational data from semi-structured spreadsheets without requiring programming expertise.  
142    FLASHRELATE introduces a domain-specific language, **FLARE**, which extends traditional regular  
143    expressions with spatial constraints to capture the geometric relationships inherent in spreadsheet  
144    layouts. Additionally, FLASHRELATE employs an algorithm that synthesizes FLARE programs

from a small number of user-provided positive and negative examples, significantly simplifying the automated data extraction process.

FLASHRELATE distinguishes itself from both DeExcelerator and XLIIndy by leveraging programming-by-example (PBE) techniques. While DeExcelerator relies on predefined heuristic rules and XLIIndy incorporates machine learning models requiring user interaction for fine-tuning, FLASHRELATE allows non-expert users to define extraction patterns through intuitive examples. This approach lowers the barrier to entry for extracting relational data from complex spreadsheet encodings, making the tool accessible to a broader range of users.

## 2.5 Senbazuru

Chen et al. [5] introduced **Senbazuru**, a prototype Spreadsheet Database Management System (SSDBMS) designed to extract relational information from a large corpus of spreadsheets. Senbazuru addresses the critical issue of integrating data across multiple spreadsheets, which often lack explicit relational metadata, thereby hindering the use of traditional relational tools for data integration and analysis.

Senbazuru comprises three primary functional components:

(1) **Search**: Utilizing a textual search-and-rank interface, Senbazuru enables users to quickly locate relevant spreadsheets within a vast corpus. The search component indexes spreadsheets using Apache Lucene, allowing for efficient retrieval based on relevance to user queries.

(2) **Extract**: The extraction pipeline in Senbazuru consists of several stages:

- **Frame Finder**: Identifies data frame structures within spreadsheets using Conditional Random Fields (CRFs) to assign semantic labels to non-empty rows, effectively detecting rectangular value regions and associated attribute regions.

- **Hierarchy Extractor**: Recovers attribute hierarchies for both left and top attribute regions. This stage also incorporates a user-interactive repair interface, allowing users to manually correct extraction errors, which the system then generalizes to similar instances using probabilistic methods.

- **Tuple Builder and Relation Constructor**: Generates relational tuples from the extracted data frames and assembles these tuples into coherent relational tables by clustering attributes and recovering column labels using external schema repositories like Freebase and YAGO.

(3) **Query**: Supports basic relational operations such as selection and join on the extracted relational tables, enabling users to perform complex data analysis tasks without needing to write SQL queries.

Senbazuru's ability to handle hierarchical spreadsheets, where attributes may span multiple rows or columns without explicit labeling, sets it apart from earlier systems like DeExcelerator and XLIIndy. By employing machine learning techniques and providing user-friendly repair interfaces, Senbazuru ensures high-quality extraction and facilitates the integration of spreadsheet data into relational databases.

## 2.6 TableSense

Dong et al. [6] developed **TableSense**, an end-to-end framework for spreadsheet table detection using Convolutional Neural Networks (CNNs). TableSense addresses the diversity of table structures and layouts by introducing a comprehensive cell featurization scheme, a Precise Bounding Box Regression (PBR) module for accurate boundary detection, and an active learning framework to efficiently build a robust training dataset.

197 While **DeExcelerator**, **XLIndy**, **FLASHRELATE**, and **Senbazuru** focus primarily on transforming  
198 spreadsheet data into relational forms through heuristic, machine learning, and programming-  
199 by-example approaches, **TableSense** specifically targets the accurate detection of table boundaries  
200 within spreadsheets using deep learning techniques. Unlike region-growth-based methods em-  
201 ployed in commodity spreadsheet tools, which often fail on complex table layouts, TableSense  
202 achieves superior precision and recall by leveraging CNNs tailored for the unique characteristics of  
203 spreadsheet data. However, TableSense focuses on table detection and visualization, allowing users  
204 to generate diagrams from the detected tables but does not provide functionality for exporting the  
205 extracted data for further analysis.

## 207 2.7 Comparison and Positioning

208 While **DeExcelerator**, **XLIndy**, **FLASHRELATE**, **Senbazuru**, and **TableSense** each offer unique  
209 approaches to spreadsheet data extraction, they share certain limitations. Many of these tools  
210 are not readily accessible: **FLASHRELATE** and **TableSense** are proprietary, and **Senbazuru**,  
211 **XLIndy**, and **DeExcelerator** are discontinued projects with limited or no source code availability.  
212 In contrast, we contribute our spreadsheet data extractor under the GNU General Public License  
213 v3.0, allowing the community to access, use, and improve the tool freely.

214 Moreover, unlike the aforementioned tools that rely on heuristics, machine learning, or AI  
215 techniques—which can introduce errors requiring users to identify and correct—we adopt a user-  
216 centric approach that gives users full control over data selection and metadata hierarchy definition.  
217 While this requires more manual input, it eliminates the uncertainty and potential inaccuracies  
218 associated with automated methods. To streamline the process and enhance efficiency, our tool  
219 includes user-friendly features such as the ability to duplicate hierarchies of columns and tables,  
220 and to move them over similar structures for reuse, reducing the need for repetitive configurations.  
221

222 By combining the strengths of manual control with enhanced user interface features and per-  
223 formance optimizations, our tool offers a robust and accessible solution for extracting relational  
224 data from complex and visually intricate spreadsheets. These enhancements not only improve  
225 performance and accuracy but also elevate the overall user experience, making our tool a valuable  
226 asset for efficient and reliable data extraction from diverse spreadsheet formats.

## 227 3 METHODOLOGY

228 In this section, we detail the design and implementation of the Spreadsheet Data Extractor (SDE),  
229 emphasizing its user-centric approach and performance optimizations. The SDE enables users to  
230 transform semi-structured spreadsheet data into structured, machine-readable formats without  
231 requiring programming expertise. We achieve this through an intuitive interface that allows for  
232 cell selection and hierarchy definition, incremental loading of worksheets, accurate rendering of  
233 cell dimensions, and optimized performance for handling large datasets by incrementally loading  
234 worksheets and by rendering only the cells that are currently visible in the view.  
235

### 236 3.1 User-Centric Data Extraction

237 The core functionality of the SDE revolves around allowing users to select cells containing data  
238 and metadata to define a data hierarchy. This process is facilitated through a graphical interface  
239 that displays the spreadsheet and allows for intuitive selection and manipulation of the selection  
240 hierarchy.

241 242 *3.1.1 Hierarchy Definition.* Users can select individual cells or ranges of cells by clicking and using  
243 shift-click for multi-selection. These selections represent either data or metadata.

The selected cells are organized into a hierarchical tree structure, where each node represents a data element, and child nodes represent nested data or metadata. This hierarchy defines how the data will be transformed into a structured format.

**3.1.2 Reusability and Efficiency.** To optimize the extraction process and reduce repetitive tasks, the SDE allows users to duplicate previously defined hierarchies and apply them to similar regions within the spreadsheet. This feature is particularly useful for spreadsheets with repeating structures, such as multiple tables with the same format.

## 3.2 Example Workflow

Consider a spreadsheet containing statistical forecasts of future nursing staff availability in Germany [12]. Figure 1 shows the SDE interface, which consists of three main components:

**Hierarchy Panel (Top Left):** Displays the hierarchy of cell selections, initially empty.

**Spreadsheet View (Top Right):** Shows the currently opened Excel file and the currently selected worksheet for cell selection.

**Output Preview (Bottom):** Provides immediate feedback on the data extraction based on current selections.



Fig. 1. The SDE Interface Overview.

**3.2.1 Selection of the First Column.** The user adds a node to the hierarchy and selects the cell containing the metadata "Nursing Staff" (Figure 2). This cell represents metadata that is common to all cells in this worksheet. Therefore, it should be selected first and should appear at the beginning of each row in the output CSV file.

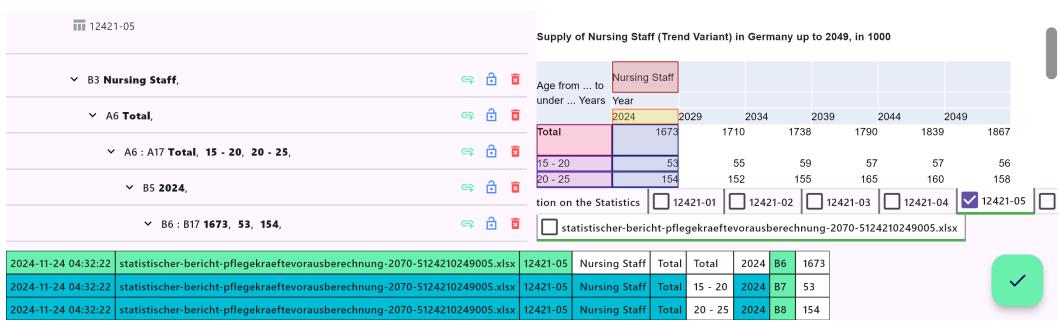


Fig. 2. Selection of the First Column Metadata

Within this node, the user adds a child node and selects the cell "*Total*", which serves as both a table header and a row label. This selection represents the table header of the first subtable. The user

295 adds another child node and selects the range of cells containing row labels (e.g., "Total", "15-20",  
 296 "20-25" and so forth) by clicking the first cell and shift-clicking the last cell.

297 A further child node is then placed under the row labels node, and the user selects the year  
 298 "2024". Subsequently, an additional child node is created beneath the year node, and the user selects  
 299 the corresponding data cells (e.g., "1673", "53", "154", etc.).

300 At this point, the hierarchy consists of five nodes, each—except the last one—containing an  
 301 embedded child node. In the upper-right portion of the interface, the chosen cells are displayed  
 302 in distinct colors corresponding to each node. The lower area shows a preview of the extracted  
 303 output. For each child node, an additional column is appended to the output. When multiple cells  
 304 are selected for a given node, their values appear as entries in new rows of the output, reflecting  
 305 the defined hierarchical structure.

306 3.2.2 *Duplicating the Column Hierarchy.* To avoid repetitive manual entry for additional years, the  
 307 user duplicates the hierarchy for "2024" and adjusts the cell selections to include data for subsequent  
 308 years (e.g., "2025", "2026") using the "Move and Duplicate" feature.

309 310 To do this, the user selects the node of the first column "2024" and right-clicks on it. A popup  
 311 opens in which the action "move and duplicate" appears, which should then be clicked, as shown  
 312 in Figure 3a.

The figure consists of two screenshots of a spreadsheet application.   
 (a) On the left, a context menu is open over a cell in the 'Year' column. The menu items include 'encapsulate', 'enter value manually', 'move and duplicate' (which is checked), 'copy cell selection', 'copy task', and 'duplicate task'. The 'move and duplicate' option is highlighted.   
 (b) On the right, a dialog box titled 'Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000' is shown. It contains a table with data for 'Nursing Staff' across various age groups ('15 - 20', '20 - 25') and years ('2024', '2029', '2034', '2039', '2044', '2049', '1867'). Below the table, there are checkboxes for 'move and duplicate' and a 'repeat' input field set to '5'. A green checkmark is next to the 'move and duplicate' checkbox. At the bottom, there are buttons for 'cancel', 'apply', and 'accept'. The 'accept' button is highlighted.

323 (a) Invoking the "move and duplicate" feature on the  
 324 325 2024 column node.

(b) Adjusting the number of repetitions to duplicate  
 the column selection.

326 Fig. 3. Utilizing the "Move and Duplicate" feature to replicate column hierarchies for additional years.  
 327

328 Subsequently, a series of buttons opens in the app bar at the top right, allowing the user to move  
 329 the cell selections of the node as well as all child nodes, as seen in Figure 3b. By pressing the button  
 330 to move the selection by one unit to the right, the next column is selected; however, this would  
 331 also deselect the first column since the selection was moved. To preserve the first column, the  
 332 "move and duplicate" checkbox can be activated. This creates the shifted selection in addition to the  
 333 original selection. However, the changes are only applied when the accept button is clicked. The  
 334 next columns could also be selected in the same way. But this can be done faster, because instead  
 335 of moving the selection and duplicating it only once, the "repeat" input field can be filled with as  
 336 many repetitions as there are columns. By entering the number 5, the selection of the first column  
 337 is shifted 5 times by one unit to the right and duplicated at each step.

338 The user reviews the selections in the spreadsheet view, where each selection is highlighted in a  
 339 different color corresponding to its node in the hierarchy. Only after the user has reviewed the  
 340

shifted and duplicated selections in the worksheet and clicked the accept button are the nodes in the hierarchy created as desired. We present the results of these runtime measurements in Figure 4.

The user reviews the selections in the spreadsheet view, where each selection is highlighted in a different color corresponding to its node in the hierarchy. Erst nachdem der Nutzer die verschobenen und duplizierten selektionen in der Worksheet ansieht überprüft hat und den Akzeptieren button geklickt hat, werden die Knoten in der Hierachie wie gewünscht angelegt. Das Ergebnis dieser Operation ist in Abbildung 4 zu sehen.

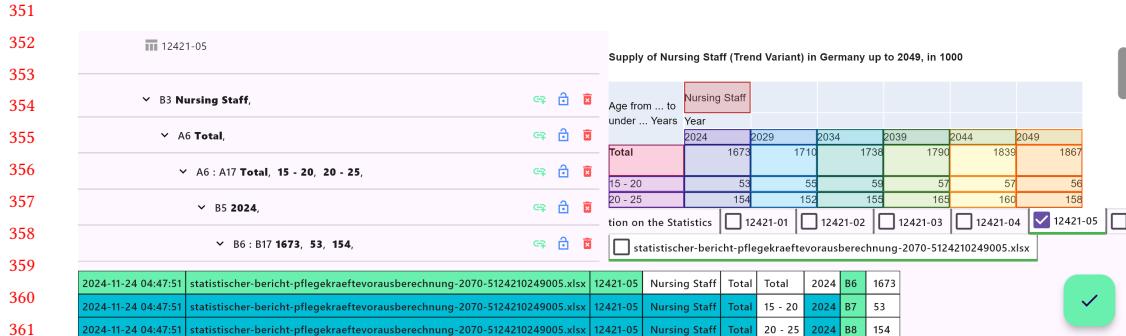


Fig. 4. Resulting Hierarchy After Move and Accept

**3.2.3 Duplicating the Table Hierarchy.** The same method that worked effectively for duplicating the columns can now be applied to the subtables, as shown in Figure 5.



Fig. 5. Selection of All Cells in the Subtables by Duplicating the Hierarchy of the First Table

By selecting the node with the value "Total" and clicking the "Move and Duplicate" button, we can apply the selection of the "Total" subtable to the other subtables. This involves shifting the table downward by as many rows as necessary to overlap with the subtable below.

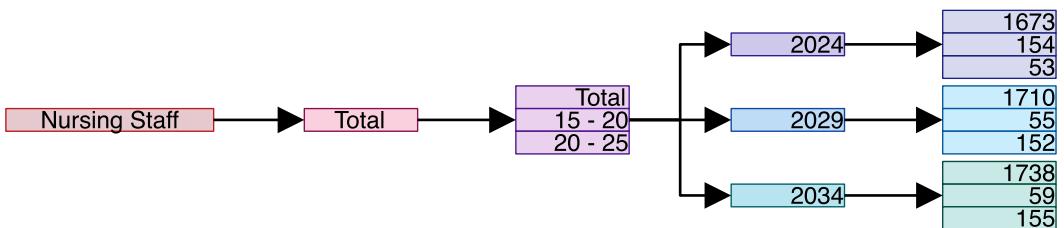
393 However, there is a minor issue: the child nodes of the "Total" node also include the column  
 394 headers. If these column headers were repeated in the subtables below, shifting the selections  
 395 downward would work without modification. Since these cells are not repeated in the subtables,  
 396 we need to prevent the column headers cells from moving during the duplication process.

397 To achieve this, we can exclude individual nodes from being moved by locking their selection.  
 398 This is done by clicking the padlock icon on the corresponding nodes, which freezes their cell  
 399 selection and keeps them fixed at their original position, regardless of other cells being moved.

400 Therefore, we identify and select the nodes containing the column headers—specifically, the  
 401 years 2024 to 2049—and lock their selection using the padlock button. By shifting the selection  
 402 downward and duplicating it, we can easily move and duplicate the cell selections for the subtables  
 403 below. By setting the number of repetitions to 2, all subtables are completely selected.

### 404 405 3.3 Cross Product Transformation

406 The graph resulting from the selected hierarchy is shown in Figure 6. To simplify the explanation,  
 407 the example is limited to the first three columns and the first three rows of the first sub-table.



417 Fig. 6. Illustration of the Cross Product Transformation Before Application

418  
 419 Once the hierarchy is defined, the **Structured Data Engine (SDE)** applies a cross-product  
 420 transformation to generate a relational format from the selection graph. This transformation  
 421 consists of two key steps:

- 422 (1) **Node Duplication:** Nodes with multiple incoming or outgoing edges (e.g., the row labels  
 423 node with the values *Total*, 15-20, 20-25) are duplicated to ensure that each edge connects to a  
 424 unique instance of the node. This adjustment replaces the original many-to-one relationships  
 425 with one-to-one mappings for each edge.
- 426 (2) **Value Replication:** Nodes containing single values (e.g., the year 2024) are replicated  
 427 to align with the number of values associated with the connected node. This ensures a  
 428 consistent structure in the relational output, maintaining alignment across all hierarchical  
 429 levels.

430  
 431 The resulting graph after the cross-product transformation is shown in Figure 7. With this  
 432 transformation applied to the selected hierarchy, the SDE generates a structured output that reflects  
 433 the original spreadsheet's hierarchical relationships, enabling users to analyze and integrate the  
 434 data effectively.

### 435 436 3.4 Incremental Loading of Worksheets

437 Opening large Excel files traditionally involves loading the entire file and all its worksheets into  
 438 memory before displaying any content. In files containing very large worksheets, this process  
 439 can take several seconds to minutes, causing significant delays for users who need to access data  
 440 quickly.

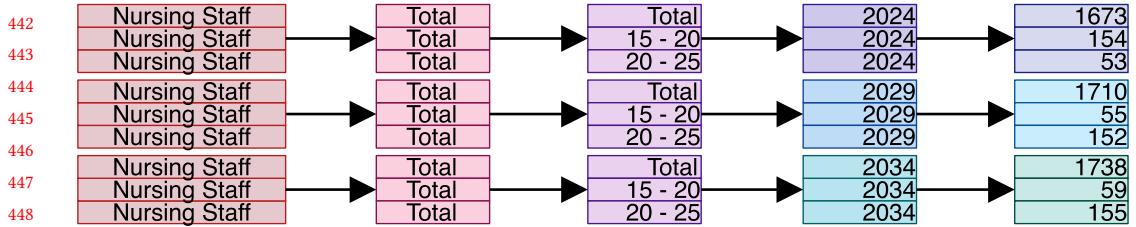


Fig. 7. Illustration of the Cross Product Transformation After Application

To facilitate efficient data extraction from multiple Excel files, we implemented a mechanism for incremental loading of worksheets within the SDE. Excel files (.xlsx format) are ZIP archives containing a collection of XML files that describe the worksheets, styles, and shared strings. Key components include:

- **xl/sharedStrings.xml**: Contains all the unique strings used across worksheets, reducing redundancy.
- **xl/styles.xml**: Defines the formatting styles for cells, including fonts, colors, and borders.
- **xl/worksheets/sheetX.xml**: Represents individual worksheets (sheet1.xml, sheet2.xml, etc.).

Our solution opens the Excel file as a ZIP archive and initially extracts only the essential metadata and shared resources required for the application to function. This initial extraction includes:

#### (1) Metadata Extraction:

We read the archive's directory to identify the contained files without decompressing them fully. This step is quick, taking only a few milliseconds, and provides information about the available worksheets and shared resources.

#### (2) Selective Extraction:

We immediately extract sharedStrings.xml and styles.xml because these files are small and contain information necessary for rendering cell content and styles across all worksheets. These files are parsed and stored in memory for quick access during rendering.

#### (3) Deferred Worksheet Loading:

The individual worksheet files (sheetX.xml) remain compressed and are loaded into memory in their binary unextracted form. They are not decompressed or parsed at this stage.

#### (4) On-Demand Parsing:

When a user accesses a specific worksheet—either by selecting it in the interface or when a unit test requires data from it—the corresponding sheetX.xml file is then decompressed and parsed. This parsing occurs in the background and is triggered only by direct user action or programmatic access to the worksheet's data.

#### (5) Memory Release:

After a worksheet has been decompressed and its XML parsed, we release the memory resources associated with the parsed data. This approach prevents excessive memory usage and ensures that the application remains responsive even when working with multiple large worksheets.

By adopting this incremental loading approach, users experience minimal wait times when opening an Excel file. The initial loading is nearly instantaneous, allowing users to begin interacting with the application without delay. This contrasts with traditional methods that require loading all worksheets upfront, leading to significant wait times for large files.

### 491 3.5 Rendering of Worksheets

492 To ensure that users can navigate worksheets without difficulty, we prioritize displaying the  
493 worksheets in a manner that closely resembles their appearance in Excel. This involves accurately  
494 rendering cell dimensions, formatting, and text behaviors.

495  
496 3.5.1 *Displaying Row Heights and Column Widths.* Our solution extracts information about column  
497 widths and row heights directly from the Excel file's XML structure. Specifically, we retrieve the  
498 column widths from the *width* attribute of the `<col>` elements and the row heights from the *ht*  
499 attribute of the `<row>` elements in the `sheetX.xml` files.

500 In Excel, column widths and row heights use units that do not directly map to pixels, requiring  
501 conversion for accurate on-screen rendering. Different scaling factors are needed for columns and  
502 rows. Through empirical testing, we derived the following scaling factors:

- 503 • **Column Widths:** Multiply the *width* attribute by 7.
- 504 • **Row Heights:** Multiply the *ht* attribute by  $\frac{4}{3}$ .

505 Despite research, we could not find official documentation explaining the rationale behind these  
506 specific scaling factors. This lack of documentation poses a challenge for accurately replicating  
507 Excel's rendering.

508  
509 3.5.2 *Cell Formatting.* Cell formatting plays a crucial role in accurately representing the appearance  
510 of worksheets. Formatting information is stored in the `styles.xml` file, where styles are defined  
511 and later referenced in the `sheetX.xml` files.

512 Each cell in the worksheet references a style index through the *s* attribute, which points to the  
513 corresponding `<xf>` (Extended Format) element within the `cel1Xfs` collection. These `<xf>` elements  
514 contain attributes such as *fontId*, *fillId*, and *borderId*, which reference specific font, fill (background),  
515 and border definitions located in the `fonts`, `fills`, and `borders` collections, respectively. By  
516 parsing these references, we can accurately apply the appropriate fonts, background colors, and  
517 border styles to each cell.

518 Through meticulous parsing and application of these formatting details, we ensure that the  
519 rendered worksheet closely mirrors the original Excel file, preserving the visual cues and aesthetics  
520 that users expect.

521  
522 3.5.3 *Handling Text Overflow.* In Excel, when the content of a cell exceeds its width, the text may  
523 overflow into adjacent empty cells, provided those cells do not contain any data. If adjacent cells  
524 are occupied, Excel truncates the overflowing text at the cell boundary. Replicating this behavior is  
525 essential for accurate rendering and user familiarity.

526 We implemented text overflow handling by checking if the adjacent cell to the right is empty  
527 before allowing text to overflow. If the adjacent cell is empty, we extend the text rendering. If the  
528 adjacent cell contains data, we truncate the text at the boundary of the original cell.

529 Figure 1 illustrates this behavior. The text "Supply of Nursing Staff ..." extends into the neighboring  
530 cell because it is empty. If not for this handling, the text would be truncated at the cell boundary,  
531 leading to incomplete data display as shown in Figure 8.

532 By accurately handling text overflow, we improve readability and maintain consistency with  
533 Excel's user interface, which is crucial for users transitioning between Excel and our tool.

### 534 3.6 Performance Optimization

535 To ensure high frame rates even with large worksheets, we optimized the Spreadsheet Data  
536 Extractor to render only the cells that are currently visible to the user. Rendering the entire  
537 worksheet, especially when it contains thousands of cells, can significantly degrade performance.

540  
 541 [To the table of c](#)  
 542  
 543  
 544 Supply of Nurs  
 545

Age from ... to under ... Years	Nursing Staff							
		Year	2024	2029	2034	2039	2044	2049
	Total	1673	1710	1738	1790	1839	1867	

552 Fig. 8. Falsche Darstellung ohne overflow von zellen mit angrenzenden zellen ohne inhalt  
 553

554  
 555  
 556 By focusing on the visible cells within the viewport, we reduce computational overhead and improve  
 557 responsiveness.  
 558

559 We achieve this optimization by utilizing the `two_dimensional_scrollables` package [10].  
 560 This package provides functionality for efficiently handling two-dimensional scrolling regions,  
 561 making it suitable for rendering large grids like spreadsheets.

562 Since we extract all column widths and row heights from the XML files, we can calculate the exact  
 563 dimensions and positions of each cell. By accumulating the widths and heights, we determine the  
 564 coordinates of each cell within the worksheet grid. These coordinates are essential for identifying  
 565 which cells fall within the current viewport and should be rendered.

566 We consider the current scroll offsets along the horizontal ( $x$ -axis) and vertical ( $y$ -axis) directions.  
 567 The viewport is defined by the width and height of the panel displaying the Excel worksheet. To  
 568 determine the visible cells, we perform the following steps:

569 • **Horizontal Visibility:**

- 570 – Sum the column widths until the accumulated sum reaches the left edge of the viewport.  
   All cells to the left are ignored.
- 571 – Continue adding column widths until the sum exceeds the right edge of the viewport.  
   Cells beyond this point are also ignored.

572 • **Vertical Visibility:**

- 573 – Sum the row heights until the accumulated sum reaches the top edge of the viewport.  
   All cells above are ignored.
- 574 – Continue adding row heights until the sum surpasses the bottom edge of the viewport.  
   Cells below this point are ignored.

575 By rendering only the cells within these boundaries, we significantly reduce the number of cells  
 576 processed at any given time.

577 The `two_dimensional_scrollables` package provides interfaces where the logic for laying out  
 578 the cells can be implemented. Parameters that describe the viewport, including the horizontal and  
 579 vertical offsets as well as the viewport height and width, are supplied by these interfaces.

580 Algorithm 1 outlines our implementation of the overridden `layoutChildSequence` function,  
 581 which is invoked by the `two_dimensional_scrollables` package to calculate and arrange the  
 582 visible cells within the viewport.

**Algorithm 1** Layout of Visible Spreadsheet Cells

---

```

589
590 1: Initialize Indices
591 2:   leadingColumnIndex  $\leftarrow$  column index corresponding to horizontalOffset
592 3:   leadingRowIndex  $\leftarrow$  row index corresponding to verticalOffset
593 4:   trailingColumnIndex  $\leftarrow$  column index corresponding to horizontalOffset + viewportWidth
594 5:   trailingRowIndex  $\leftarrow$  row index corresponding to verticalOffset + viewportHeight
595
596 6: Calculate Initial Offsets
597 7:   leadingColumnOffset  $\leftarrow$  sum of widths from the first column up to leadingColumnIndex
598 8:   leadingRowOffset  $\leftarrow$  sum of heights from the first row up to leadingRowIndex
599 9:   horizontalLayoutOffset  $\leftarrow$  leadingColumnOffset – horizontalOffset
600 10: for each columnIndex from leadingColumnIndex to trailingColumnIndex do
601 11:   verticalLayoutOffset  $\leftarrow$  leadingRowOffset – verticalOffset
602 12:   for each rowIndex from leadingRowIndex to trailingRowIndex do
603 13:     cell  $\leftarrow$  build or retrieve the cell at (columnIndex, rowIndex)
604 14:     Layout cell at position (horizontalLayoutOffset, verticalLayoutOffset)
605 15:     if a custom height is defined for row rowIndex then
606 16:       verticalLayoutOffset  $\leftarrow$  verticalLayoutOffset + height of row rowIndex
607 17:     else
608 18:       verticalLayoutOffset  $\leftarrow$  verticalLayoutOffset + defaultRowHeight
609 19:     end if
610 20:   end for
611 21:   columnWidth  $\leftarrow$  width of column columnIndex
612 22:   horizontalLayoutOffset  $\leftarrow$  horizontalLayoutOffset + columnWidth
613 23: end for
614
615

```

---

616 By applying this method, we render only the cells necessary for the current view, thereby  
617 optimizing performance and ensuring smooth user interactions even with large and complex  
618 worksheets.

619

## 620 4 EVALUATION

621 The fundamental approach of the Spreadsheet Data Extractor, based on the converter by Aue et  
622 al. [1], upon which we build, remains unchanged. The effectiveness of this approach has already  
623 been investigated. Aue et al. evaluated the extraction of data from over 500 Excel files. The time  
624 required for each file was determined from a sample of 331 processed Excel files comprising 3,093  
625 worksheets. On average, student assistants needed 15 minutes per file and 95 seconds per worksheet.  
626

627 Our focus is on improving the user experience and optimizing the performance of the Spreadsheet  
628 Data Extractor. We enhanced the user experience by displaying the Excel worksheets similarly  
629 to how they appear in Excel and by reducing the number of required user interactions through  
630 the integration of the selection hierarchy, worksheet view, and output preview into a single  
631 interface. Performance was further improved by implementing incremental loading of Excel files  
632 and rendering only the visible cells.

633

### 634 4.1 Acceleration When Opening Files

635 To evaluate the performance improvements when opening Excel files, we conducted a series of tests  
636 using a large Excel file. We downloaded the entire collection of Excel files from the German Federal  
637

Statistical Office (Destatis)<sup>1</sup>, which provides extensive statistical data across various domains. From this collection, we identified the largest file [13], which has a compressed size of 87 MB and an uncompressed size of 911 MB.

We performed three sets of tests to compare the performance of different methods for opening and reading data from this Excel file:

- (1) **VBA Script in Excel:** We wrote a VBA script that opens the Excel file and reads specific values from a worksheet. This script simulates how users might interact with the file using Excel's built-in capabilities. We ran this script 10 times, measuring the time required to open the file and read the values in each iteration.
- (2) **Spreadsheet Data Extractor:** We developed an equivalent unit test using the functions implemented in our Spreadsheet Data Extractor to open the same file and read the same cells. This test aimed to assess the performance of our tool under the same conditions. We repeated this test 10 times, recording the runtime for each execution.
- (3) **excel Package:** We tested the excel package [?] used by Aue et al. [1], creating a unit test that attempts to read the same cells from the file. This test was intended to benchmark the performance of the prior solution. However, due to limitations with the package, we encountered an out-of-memory exception during the second run; therefore, only the time from the first run was recorded.

All tests were conducted on a machine with the following specifications: Intel Core i5-10210U quad-core CPU at 1.60 GHz, 16 GB RAM, and a solid-state drive (SSD), running Windows 10 Pro. The version of Microsoft Excel used was Microsoft Office LTSC Professional Plus 2021.

The results of these runtime measurements are presented in Figure 9.

The Spreadsheet Data Extractor opened the worksheet with a median time of 120 milliseconds and an average time of 178 milliseconds. The first run took 668 milliseconds, possibly because the file was not yet cached and had to be loaded from the disk.

In contrast, Excel opened the worksheet with a median time of 40.281 seconds and an average time of 41.138 seconds.

The Dart excel package [?] used in the previous work took 13 minutes and 15 seconds to open the worksheet in the first run. The other nine runs could not be completed because an out-of-memory exception was thrown during the second run.

These results demonstrate that the Spreadsheet Data Extractor opens worksheets over two orders of magnitude faster than Excel and nearly four orders of magnitude faster than the Dart excel package used in prior work.

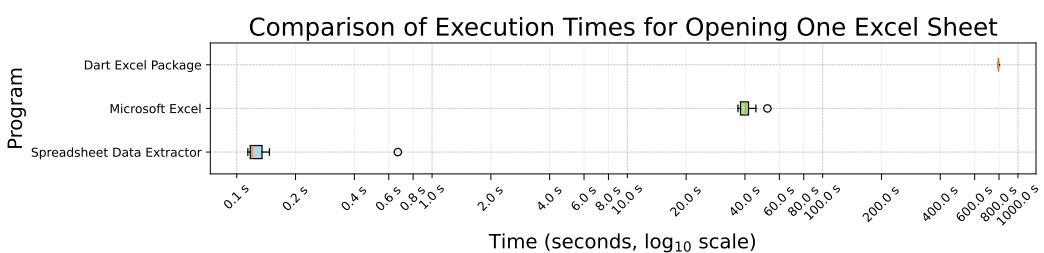


Fig. 9. Boxplot of Worksheet Opening Times Comparing the Spreadsheet Data Extractor, Microsoft Excel, and the Dart excel Package

<sup>1</sup><https://www.destatis.de>

## 687 5 FUTURE WORK

688 We plan to continue improving the Spreadsheet Data Extractor by implementing new features that  
 689 enhance the user experience and address current limitations. One such improvement is the correct  
 690 rendering of text aligned using Excel's "Center Across Selection" horizontal alignment.

691 In parallel, we intend to test the tool on additional datasets to further evaluate its effectiveness  
 692 and efficiency.

### 694 5.1 Implementing "Center Across Selection"

695 Although we can analyze cell merging—a commonly used feature in Excel to center text across  
 696 multiple cells—this worksheet does not use this visual formatting feature through cell merging.  
 697 Instead, the first cell has a horizontal alignment set to centerContinuous. Our previous attempts to  
 698 determine the cell coordinates over which this horizontal centering extends have been unsuccessful.  
 699 Further testing and research are necessary to understand how such centered texts are stored in the  
 700 XML structure of the Excel file.

701 [To the table of contents](#)

## 704 705 Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000

707 Age from ... 708 to under ... 709 Years	Nursing Staff					
	Year					
	2024	2029	2034	2039	2044	2049
710	711	712 Fig. 10. Illustration of the text alignment "Center Across Selection" in Excel	713	714	715	716

## 716 5.2 Evaluation on Real-World Data

717 To date, the base version of the tool has been tested on a dataset from the Agricultural Structure  
 718 Survey on land use and livestock in Germany for 2020. It would be valuable to test the new version  
 719 of the tool on data from before 2020 to assess whether the improvements we have made enhance  
 the tool's effectiveness.

720 We plan to utilize the timestamps documented in the output CSV files to compare them with  
 721 new timestamps obtained from re-testing. This comparison will help us determine whether the  
 722 student assistants can work faster with the new version of the tool.

## 723 6 CONCLUSION

724 In this paper, we presented the Spreadsheet Data Extractor, an enhanced tool that builds upon the  
 725 foundational work of Alexander Aue et al. [1]. By improving upon their existing approach, we  
 726 addressed key limitations and introduced significant enhancements to both performance and user  
 727 experience.

728 Our contributions include implementing incremental loading of worksheets, accurate rendering  
 729 of cell dimensions by parsing XML data, and optimizing the rendering engine to draw only the  
 730 visible cells. These improvements enable the tool to open large Excel files over two orders of  
 731 magnitude faster than Microsoft Excel itself and nearly four orders of magnitude faster than the  
 732 previous implementation by Aue et al. We also integrated the selection hierarchy, worksheet view,  
 733 and output preview into a unified interface, streamlining the data extraction process.

736 By adopting a user-centric approach that gives users full control over data selection and metadata  
 737 hierarchy definition without requiring programming knowledge, we provide a robust and accessible  
 738 solution for data extraction. Our tool offers user-friendly features such as the ability to duplicate  
 739 hierarchies of columns and tables and to move them over similar structures for reuse, reducing the  
 740 need for repetitive configurations.

741 By contributing our improved Spreadsheet Data Extractor under the GNU General Public License  
 742 v3.0, we enable the community to access, use, and enhance the tool freely, fostering collaboration  
 743 and further innovation. By combining the strengths of the original approach with our enhancements  
 744 in user interface and performance optimizations, our tool significantly improves the efficiency and  
 745 reliability of data extraction from diverse and complex spreadsheet formats.

## 746 REFERENCES

- 748 [1] Andrea Ackermann Alexander Aue. 2024. Converting data organised for visual perception into machine-readable  
 749 formats. In *44. GIL-Jahrestagung, Biodiversität fördern durch digitale Landwirtschaft*. Gesellschaft für Informatik eV,  
 750 179–184.
- 751 [2] Anonymous. [n.d.]. Spreadsheet Data Extractor (SDE). [https://anonymous.4open.science/r/spreadsheet\\_data\\_extractor-13BD/README.md](https://anonymous.4open.science/r/spreadsheet_data_extractor-13BD/README.md). Accessed: 2024-12-08.
- 752 [3] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from  
 753 semi-structured spreadsheets using examples. *ACM SIGPLAN Notices* 50, 6 (2015), 218–228.
- 754 [4] D.J. Berndt, J.W. Fisher, A.R. Hevner, and J. Studnicki. 2001. Healthcare data warehousing and quality assurance.  
 755 *Computer* 34, 12 (2001), 56–65. <https://doi.org/10.1109/2.970578>
- 756 [5] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. 2013. Senbazuru: A prototype spreadsheet  
 757 database management system. *Proceedings of the VLDB Endowment* 6, 12, 1202–1205.
- 758 [6] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. Tablesense: Spreadsheet table detection with  
 759 convolutional neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 69–76.
- 760 [7] Angus Dunn. 2010. Spreadsheets-the Good, the Bad and the Downright Ugly. *arXiv preprint arXiv:1009.5705* (2010).
- 761 [8] Julian Eberius, Christopher Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. 2013.  
 762 DeExcelerator: a framework for extracting relational data from partially structured documents. In *Proceedings of the  
 763 22nd ACM international conference on Information & Knowledge Management*. 2477–2480.
- 764 [9] Elvis Koci, Dana Kuban, Nico Luettig, Dominik Olwig, Maik Thiele, Julius Gonsior, Wolfgang Lehner, and Oscar  
 765 Romero. 2019. Xlindy: Interactive recognition and information extraction in spreadsheets. In *Proceedings of the ACM  
 766 Symposium on Document Engineering 2019*. 1–4.
- 767 [10] Kate Lovett. 2023. two\_dimensional\_scrollables package - Commit 4c16f3e. <https://github.com/flutter/packages/commit/4c16f3ef40333aa0aebe8a1e46ef7b9fef9a1c1f> Accessed: 2023-08-17.
- 768 [11] Gursharan Singh, Leah Findlater, Kentaro Toyama, Scott Helmer, Rikin Gandhi, and Ravin Balakrishnan. 2009. Nu-  
 769 matic paper forms for NGOs. In *2009 International Conference on Information and Communication Technologies and  
 770 Development (ICTD)*. IEEE, 406–416.
- 771 [12] Statistisches Bundesamt (Destatis). 2024. *Statistischer Bericht - Pflegekräftevorausberechnung - 2024 bis 2070*. Technical  
 772 Report. Statistisches Bundesamt, Wiesbaden, Germany. <https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Bevoelkerung/Bevoelkerungsvorausberechnung/Publikationen/Downloads-Vorausberechnung/statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.html> Statistical Report - Projection of Nursing Staff - 2024 to 2070.
- 773 [13] Statistisches Bundesamt (Destatis). 2024. Statistischer Bericht: Rechnungsergebnis der Kernhaushalte der Gemeinden.  
 774 <https://www.destatis.de/DE/Themen/Staat/Oeffentliche-Finanzen/Ausgaben-Einnahmen/Publikationen/Downloads-Ausgaben-und-Einnahmen/statistischer-bericht-rechnungsergebnis-kernhaushalt-gemeinden-2140331217005.html> Accessed: 2024-11-29.
- 775 [14] Hannah West and Gina Green. 2008. Because excel will mind me! the state of constituent data management in small  
 776 nonprofit organizations. In *Proceedings of the Fourteenth Americas Conference on Information Systems*. Association for  
 777 Information Systems, AIS Electronic Library (AISel). <https://aisel.aisnet.org/amcis2008/336>

779 Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009