Integrating traditional web crawling with declarative XML output

Rashid Turgunbaev
Kokand State University

Abstract: *The proliferation of dynamic web content and complex site architectures presents significant challenges for automated web indexing and search engine optimization. While the Sitemap XML protocol offers a standardized format for declaring a website's structure, the automated generation of accurate and comprehensive sitemaps remains a non-trivial computational task. This article presents a detailed analysis of a novel software artifact, a Sitemap Crawler, which synthesizes traditional breadth-first web crawling techniques with robust XML generation to produce protocol-compliant sitemaps. We examine the architectural decisions, data structures, and algorithms embodied in its Java implementation, focusing on its dual nature as both a graphical user interface application and a background processing engine. The crawler's innovative use of batched disk I/O for memory management, its heuristic for polite crawling, and its sophisticated URL filtering mechanism are critically evaluated. This analysis contributes to the discourse on practical web data extraction tools by demonstrating a functional model that balances efficiency, robustness, and user accessibility in the domain of automated sitemap creation.*
Keywords: *sitemap generation, web crawling algorithm, batched I/O management, xml protocol compliance, polite crawling, java swing application*

Introduction

The structural discovery of web resources is a cornerstone of an open and indexable internet. The Sitemaps protocol, an open standard adopted by major search engines, allows webmasters to explicitly list important pages, providing metadata such as update frequency and priority. However, manual sitemap creation is infeasible for large, evolving websites. Automated generators must therefore intelligently traverse a site, discerning navigational links from irrelevant ones (e.g., to media files, external sites, or functional elements), and serialize the results into a valid XML structure. The presented Sitemap Crawler addresses this need not as a command-line utility but as an accessible desktop application, embedding complex crawling logic within a responsive Swing GUI. This article deconstructs its implementation, revealing a design that thoughtfully addresses core challenges in web crawling: state management, politeness, scalability, and output standardization. The tool does not parse existing sitemaps but actively constructs them through exploration, representing a proactive rather than a declarative approach to site mapping.

Architectural Overview

The system is architecturally partitioned into two distinct layers: a presentation layer built with Java Swing and a core crawling engine. The NewCrawler class extends JFrame, serving as the primary controller and view. Upon user initiation, it spawns a dedicated thread for the crawling process, ensuring the GUI remains responsive - a critical design pattern for long-running I/O operations. The core logic is encapsulated within the inner Scan class, which implements Runnable. This separation of concerns isolates the inherently blocking network and file operations from the event dispatch thread of the GUI, preventing application freezing. The crawling engine itself manages three primary data structures: a Queue<String> (urlQueue) for managing the frontier of unexplored URLs in a breadth-first manner, a Set<String> (visitedUrls) for global duplicate prevention, and a temporary file system store for scalability. This hybrid in-memory and on-disk state management is a defining

feature of the architecture, enabling the tool to handle websites of varying sizes without succumbing to memory exhaustion.

The Crawling Algorithm

The crawler implements a classic breadth-first search algorithm over the graph defined by hyperlinks within a domain. The algorithm begins by seeding the queue with the user-provided baseUrl. The main loop dequeues a URL, fetches its HTML document using the JSoup library, extracts all anchor (<a>) tags, and enqueues any novel, valid URLs found. The BFS strategy is well-suited for sitemap generation as it tends to discover pages in order of their linkage distance from the homepage, often correlating with structural importance. A significant algorithmic enhancement is the integrated 100-millisecond delay (Thread.sleep(100)) within the processing loop. This implements a politeness policy, reducing server load and minimizing the risk of being blocked for aggressive requesting, an often-overlooked aspect of ethical web crawling.

The isValidUrl method acts as a crucial filter, defining the edge criteria for the traversal graph. It employs a multi-pronged heuristic to exclude non-essential resources. It enforces scope by requiring URLs to start with the baseUrl (preventing crawler drift to external sites). It ignores fragment identifiers (strings following a #). It uses a regular expression to filter out common file extensions for binaries, media, and documents (e.g., .jpg, .pdf). Furthermore, it explicitly filters out URLs containing protocols like mailto:, tel:, and javascript:, and heuristically ignores common feed URLs. This filter is instrumental in shaping the final sitemap to contain only HTML pages intended for human consumption, aligning the crawl with the semantic goal of a page sitemap.

Scalability and State Management Through Batched I/O

A central contribution of this design is its approach to memory scalability. For large websites, storing hundreds of thousands of URLs in a HashSet can strain the Java heap. The Scan class ingeniously addresses this by introducing a batched flushing mechanism. It maintains a temporary batchUrls set and a corresponding counter. When the batch size reaches a predefined FLUSH_THRESHOLD (100 URLs), the entire batch is appended to a temporary text file on disk, and the in-memory batch is cleared. A global allVisitedUrls set is kept to ensure cross-batch duplicate detection, but this set only contains the URLs of the current run; the primary persistent storage becomes the file system.

Upon completion of the crawl, the readUrlsFromTempFile method reconstitutes the complete, deduplicated set by reading all lines from the temporary file back into the master visitedUrls set. This design represents a time-space trade-off, exchanging increased disk I/O operations for drastically reduced memory footprint. It allows the application to theoretically crawl sites of nearly unlimited size, constrained only by disk space, which is a more abundant resource than RAM in typical desktop environments. The temporary file is properly cleaned up after sitemap generation, demonstrating responsible resource management.

XML Generation and Protocol Compliance

The generateSitemap method transitions from data collection to formal declaration. It utilizes the standard Java Document Object Model (DOM) API to construct a well-formed XML document. It correctly establishes the root <urlset> element with the mandatory XML namespace xmlns="http://www.sitemaps.org/schemas/sitemap/0.9". For each discovered URL, it creates a <url> element containing child nodes for <loc>, <lastmod>, <priority>, and <changefreq>.

The implementation makes specific semantic choices. The <lastmod> field is populated with the current timestamp in ISO 8601 format at the moment of generation, a pragmatic, though not perfectly accurate, approximation. The <priority> field uses a simple heuristic: the base URL receives a priority of "1.0", while all others receive "0.5". The <changefreq> is hard-coded to "weekly". While these static assignments are a simplification - advanced crawlers might infer priority from link centrality

or change frequency from historical data - they satisfy the protocol's requirements and provide a reasonable default structure. The use of a Transformer with explicit output properties ensures the final sitemap.xml file is human-readable, indented, and UTF-8 encoded.

User Interaction and Error Resilience

The GUI provides a straightforward interface: a text field for the base URL and a "Generate" button. The application performs basic validation, ensuring the URL uses HTTP/HTTPS and normalizes it with a trailing slash. The statusArea text area, using a monospaced font, serves as a real-time log, providing vital feedback on the crawl's progress, including the current URL being scanned, flush notifications, and error messages. This transparency is essential for user trust, especially when operations take considerable time.

Error handling is integrated throughout the core Scan class. Network timeouts and I/O exceptions during page fetching are caught and logged to the status area without halting the entire crawl. Exceptions during XML parsing or file operations are similarly handled. The try-catch blocks within the run() method ensure that a failure in one part of the process does not crash the thread but allows for graceful error reporting and the re-enabling of the generate button. This defensive programming approach enhances the tool's robustness for dealing with the unpredictable nature of network resources.

Comparative Analysis and Discussion

This Sitemap Crawler occupies a distinct niche. Unlike server-side scripts or plugins (e.g., those for WordPress) that generate sitemaps from a content management system's database, this tool performs an external, content-agnostic crawl. This makes it universally applicable to any publicly accessible website, regardless of its underlying technology. Compared to distributed, high-performance crawlers like Apache Nutch, it is a single-threaded, desktop-focused tool prioritizing simplicity and correctness over raw speed and scale.

Its most significant architectural decision - the batched file I/O - distinguishes it from many in-memory crawlers found in tutorials. This elevates it from a pedagogical example to a potentially practical tool for larger sites. However, limitations are evident. The crawl is single-threaded and sequential, making it slow for very large sites. It does not respect the robots.txt exclusion protocol, a critical component of compliant web crawling. The heuristics in isValidUrl, while effective, are static and might require modification for non-standard site structures. Furthermore, it cannot execute JavaScript, rendering it blind to links generated dynamically by client-side scripts, a growing challenge in the modern web.

Conclusion and Future Work

The analyzed Sitemap Crawler presents a coherent and effective synthesis of classic web crawling algorithms and XML document production within an accessible desktop application. Its design demonstrates thoughtful consideration of real-world constraints, notably through its politeness delay, batched disk I/O for scalability, and comprehensive URL filtering. It serves as a functional blueprint for understanding the core components of a dedicated sitemap generation tool.

Future iterations of this system could explore several enhancements to increase its power and compliance. Integrating a robots.txt parser would align it with web standards. Implementing a configurable thread pool would parallelize HTTP requests, significantly improving performance on bandwidth-limited crawls. The filtering heuristics could be made user-configurable via the GUI. Exploring headless browser integration (e.g., via Selenium) would enable the rendering of JavaScript-dependent content. Finally, incorporating the ability to read and merge with existing sitemap files would allow for incremental updates rather than full recrawls. Despite these potential advancements, the current implementation stands as a testament to a viable and instructive approach to automating

a key task in webmastery and SEO, bridging the gap between theoretical graph traversal and practical, standards-based output generation.

References

1. Newman, M. W., & Landay, J. A. (2000, August). Sitemaps, storyboards, and specifications: A sketch of web site design practice. In Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques (pp. 263-274).
2. Manhas, J. (2014). Comparative study of website sitemap feature as design issue in various websites. IJEM-International Journal of Engineering and Manufacturing (IJEM), 4, 22.
3. Bernard, M. (1999). Sitemap Design: Alphabetical or Categorical?. Usability news, 1(2).
4. Ceci, M., & Lanotte, P. F. (2021). Closed sequential pattern mining for sitemap generation. World Wide Web, 24(1), 175-203.
5. Khodaparasti, S., & Ahmadzadeh, M. (2011, July). A New Method for Designing a Sitemap. In International Conference on Human-Computer Interaction (pp. 580-583). Berlin, Heidelberg: Springer Berlin Heidelberg.
6. Manhas, J. (2015, March). Design and development of automated tool to study sitemap as design issue in Websites. In 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 514-518). IEEE.
7. Pilgrim, C. (2007). Trends in sitemap designs: a taxonomy and survey.