Search-Based Merge Conflict Resolution

Heleno de Souza Campos Junior
Instituto de Computação, Universidade Federal Fluminense
Niterói, RJ, Brazil
helenocampos@id.uff.br

Leonardo Gresta Paulino Murta
Instituto de Computação, Universidade Federal Fluminense
Niterói, RJ, Brazil
leomurta@ic.uff.br

Abstract

Merge conflicts are a pervasive bottleneck in collaborative software development. Automated approaches fall into two categories: *merge approaches*, which use language syntax to reduce the number of conflicts produced during the merge operation, and *conflict resolution approaches*, which receive an existing textual conflict as input and generate a valid resolution. Conflict resolution remains a challenging open problem; existing learning-based tools achieve strong results but require substantial training data and computational infrastructure. This paper presents a doctoral dissertation that applies search-based software engineering to conflict resolution. Targeting the 87% of resolutions that are formed by using lines already present in the conflicting versions [17], the dissertation delivers four integrated contributions: (i) MESTRE, a machine-learning classifier that predicts conflict resolution strategies with 80.8% accuracy and correctly resolves 70.5% of conflicts automatically; (ii) an empirical feasibility study revealing that structural properties of real resolutions reduce the search space by 89.2% while preserving 98.6% coverage; (iii) a lightweight evaluation function based on textual similarity to the conflicting parents, achieving Spearman correlation $\rho = 0.791$ with resolution quality; and (iv) SBCR, a search-based resolution tool achieving median resolution similarity of 86.5% and exact matches in 25.2% of cases [5]; its empirical evaluation across more than 50,000 conflict chunks [11] further shows that SBCR is complementary to MergeGen [15], the current state-of-the-art learning-based approach, motivating hybrid resolution strategies.

Keywords

Software merge, Conflict resolution, Search-based software engineering, Machine learning, Empirical software engineering

1 Introduction

Version control systems are central to collaborative software development, enabling teams to work concurrently on the same codebase through branching and merging workflows [22]. While branching allows independent progress, merging inevitably requires reconciling divergent changes made by different developers.

When two developers modify the same region of a file, the merge tool cannot determine the correct integration automatically, producing a textual merge conflict that must be resolved manually [20]. Conflicts are frequent and represent a recognized bottleneck in collaborative development [19, 21]. An important characteristic of these conflicts, reported by Ghiotto et al. [17] in a study of 2,731 open-source Java projects, is that 87% of all resolutions are formed by using lines already present in the conflicting versions.

Automated approaches to merge conflicts fall into two distinct categories. *Merge approaches*, such as structured and semistructured merge [2, 3, 12], operate during the merge process itself: by exploiting language syntax and parse-tree information, they reduce the number of conflicts that arise. However, they require language-specific grammars, incur performance overhead, and do not resolve conflicts that persist after merging. *Conflict resolution approaches* take an existing textual conflict as input and attempt to produce a valid resolution. Learning-based methods [14, 15, 25, 26] leverage large corpora of historical conflict–resolution pairs and achieve strong results, but require substantial training data, GPU infrastructure, and can struggle with conflict patterns absent from the training set. Search-based software engineering (SBSE) [18], a paradigm that frames problems as optimization over a candidate solution space, has not been systematically applied to conflict resolution.

This dissertation addresses this gap by focusing on the 87% of resolutions formed by using lines already present in the conflicting versions [17]. We combine two complementary approaches: a machine-learning classifier (MESTRE) that handles directly-resolvable conflicts (where the resolution keeps either conflicting version verbatim or concatenates them), and a search-based tool (SBCR) that handles the combination-based subset by searching over candidate combinations of lines from the two versions. Three central obstacles motivate the research: (i) the lack of a strategy classifier to route each conflict to the appropriate resolver; (ii) the combinatorial explosion in the number of candidate line orderings for the search-based path; and (iii) the absence of a lightweight evaluation function that can score candidates without expensive compilation or testing. The central thesis is that *it is feasible to automatically resolve a sizeable fraction of merge conflicts by predicting the resolution strategy and, when applicable, combining the conflicting lines guided by a lightweight evaluation function based on similarity to the conflicting parents.* Obstacles (i)–(iii) motivate three foundational contributions, and a fourth contribution integrates them into a single resolution tool. The two approaches are thus unpacked into four integrated contributions, detailed in Sections 3–6: MESTRE (the strategy classifier, addressing (i)), an empirical feasibility study (addressing (ii)), a lightweight parent-similarity evaluation function (addressing (iii)), and SBCR (the search-based tool that integrates them).

These contributions yield the following headline results. MESTRE predicts resolution strategies with 80.8% accuracy and correctly resolves 70.5% of all conflicts automatically (a figure that combines per-strategy accuracy with the distribution of strategies in the dataset). The feasibility study reveals structural properties of real resolutions that reduce the search space by 89.2% while preserving 98.6% coverage. The evaluation function achieves Spearman correlation $\rho = 0.791$ with resolution quality. SBCR achieves a median resolution similarity of 86.5% with exact matches in 25.2% of cases [5]; its empirical comparison with MergeGen [15], the current state-of-the-art learning-based tool, reveals complementary strengths that motivate hybrid resolution strategies.

The remainder of this paper is organized as follows. Section 2 introduces key concepts and a running example. Sections 3–6 detail the four contributions. Section 7 discusses implications for practitioners and researchers. Section 8 addresses threats to validity. Section 9 surveys related work. Section 10 concludes with future directions.

2 Background

A merge scenario involves a common base version and two diverging branches (V1 and V2) that must be integrated. The merge tool produces a combined result, and a merge conflict is a region where the tool cannot determine the correct integration automatically [20]. Phillips et al. [22] show that conflicts are a persistent source of friction in practice, and their frequency increases with team size and branch longevity.

Key concepts. A *conflict chunk* (or simply *chunk*) is the minimal conflicting region, bounded by Git's <<<<<<<, =======, and >>>>>>> markers. Each chunk presents two versions: *V1*, the lines from the first branch, and *V2*, the lines from the second branch. The task of *conflict resolution* is to replace the chunk with a sequence of lines that correctly integrates both contributions. Ghiotto et al. [17] originally categorize resolutions into six strategies; we focus on the five that occur with non-negligible frequency: keep *V1* verbatim; keep *V2* verbatim; concatenate the two versions (*CC*); combine a subset of lines from both versions in a custom order (*CB*); or introduce entirely new code (*NC*). The sixth strategy (*None*, where the resolution discards both versions entirely, distinct from *NC*, which writes new code instead) is too rare in practice and is omitted hereafter. The first three strategies are *direct*: once predicted, the resolution can be produced mechanically without further search. CB and NC require additional effort: CB through search over orderings of subsets of lines, NC through code generation or manual editing. The dissertation focuses on the 87% of resolutions formed by using lines already present in the conflicting versions (i.e., V1, V2, CC, and CB), leaving NC out of scope (MESTRE still predicts NC as a class so that such chunks are routed to manual resolution, but no automatic code generation is attempted).

Running example. Figure 1 shows a real conflict from the `hyphanet/fred` project [5] that we revisit throughout the paper. In V1, a developer adds an `AlertElement` and invokes `getInfobox` with two arguments; in V2, another developer creates a summary element and extends the same `getInfobox` call with extra display parameters (`"queue-empty"`, `true`). Keeping V1 alone loses the extended display; keeping V2 alone loses the alert. A simple concatenation duplicates the `getInfobox` assignment. The developer's resolution is CB: combine line 1 of V1 (the alert element) with line 2 of V2 (the extended `getInfobox` call), capturing both contributions. Finding this combination automatically requires searching over candidate line orderings: the central challenge the dissertation addresses.

3 Predicting Resolution Strategies

Our first contribution is MESTRE [16], the *MErge STrategy REcommender*: a machine-learning (ML) classifier that predicts the resolution strategy a developer will apply to each conflict chunk. Figure 2 situates this stage within the complete dissertation pipeline, which the following sections develop in detail.

Resolution strategies. Building on the taxonomy from Ghiotto et al. [17], MESTRE further splits CC into two distinct strategies depending on the concatenation order: *CC12* (concatenate V1 followed by V2) and *CC21* (concatenate V2 followed by V1). MESTRE therefore predicts among six classes: V1, V2, CC12, CC21, CB, and NC. The first four are *direct*: once predicted, the resolution can be applied mechanically without any search. CB requires additional effort and is handled by the subsequent contributions; NC requires code generation or manual editing and is left out of scope.

Dataset and approach. Because MESTRE requires per-project training (described below), we prioritized chunks-per-project over project diversity: we collected 52,498 conflict chunks from 20 Java open-source projects hosted on GitHub. The complementary studies in Sections 4 and 5 take the opposite trade-off, sampling CB-only conflicts across hundreds of projects to maximize cross-project diversity. MESTRE extracts 34 features at three scopes: the conflict chunk itself (e.g., number of lines in V1 and V2, presence of specific Java language constructs such as import statements and method invocations), the surrounding merge context (e.g., lines of code (LOC) added and removed per branch, frequency of keywords in commit messages), and the file context (e.g., file size and cyclomatic complexity). We trained and tuned three classifier families (Decision Tree, Random Forest, and XGBoost) using 10-fold stratified cross-validation. To capture project-specific coding conventions, models are trained *per-project* rather than across projects.

Results. Random Forest achieved the highest average accuracy of **80.8%**, ranked first in 19 of 20 projects, compared to a majority-class baseline of 57.6%, a **+54.8%** normalized improvement. It was therefore selected as the classification algorithm for MESTRE. MESTRE correctly resolves **70.5%** of conflicts automatically: for each direct strategy (V1, V2, CC12, CC21), whenever MESTRE correctly predicts it, the resolution can be applied mechanically. The 70.5% figure reflects both MESTRE's per-strategy accuracy and the distribution of strategies across the dataset, i.e., it accounts for how often each direct strategy appears and how accurately MESTRE predicts each one. Conflicts predicted as CB motivate the search-based approach in the following sections. Returning to the running example (Figure 1): neither V1 alone, nor V2 alone, nor any concatenation matches the developer's resolution. If MESTRE correctly predicts the CB strategy, the chunk is routed to the search-based path of the pipeline (Figure 2).

4 Feasibility of Combination-Based Resolutions

Before building a search-based tool, we need to understand whether real combination-based resolutions are tractable. This contribution [4, 6][1] characterizes the structure of such resolutions and derives constraints that reduce the candidate search space.

Dataset. We collected 10,177 conflict chunks from 1,076 Java open-source projects, all resolved using the combination-based (CB) strategy.

Research questions and findings. *RQ1 (Size)*: the median conflict chunk has 6 lines and the median resolution has 3 lines; 75% of all merge commits contain at most 8 conflict chunks. These small

[1]The initial work received the Distinguished Paper Award at SBES 2022.

Conflict (`QueueToadlet.java`, project `hyphanet/fred`, commit `e65a609` [5]):

```
<<<<<< V1
contentNode.addChild(new AlertElement(ctx));
HTMLNode infoboxContent = pageMaker.getInfobox("infobox-information",
        L10n.getString("QueueToadlet.globalQueueIsEmpty"), contentNode);
======
contentNode.addChild(core.alerts.createSummary());
HTMLNode infoboxContent = pageMaker.getInfobox("infobox-information",
        L10n.getString("QueueToadlet.globalQueueIsEmpty"), contentNode,
        "queue-empty", true);
>>>>>> V2
```

CB resolution (line 1 of V1 + line 2 of V2):

```
contentNode.addChild(new AlertElement(ctx));
HTMLNode infoboxContent = pageMaker.getInfobox("infobox-information",
        L10n.getString("QueueToadlet.globalQueueIsEmpty"), contentNode,
        "queue-empty", true);
```

Figure 1: Running example: real conflict from `QueueToadlet.java` (`hyphanet/fred`, commit `e65a609`) [5]. V1 uses `AlertElement` and a shorter `getInfobox` call; V2 uses `createSummary` and extends the call with display parameters. The CB resolution combines line 1 of V1 (alert element) with line 2 of V2 (extended `getInfobox`), capturing both contributions.

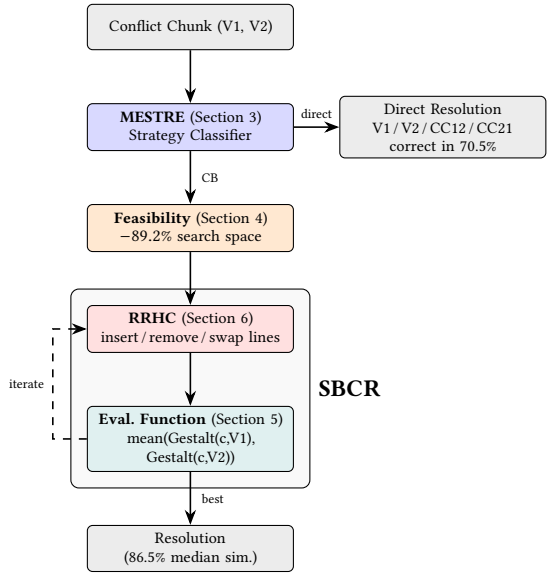

Figure 2: Dissertation pipeline. MESTRE (Section 3) predicts the resolution strategy; predicted direct strategies (V1/V2/CC12/CC21) are applied immediately, yielding a correct auto-resolution in 70.5% of conflicts overall; chunks predicted as CB are handled by SBCR through guided local search over a feasibility-constrained search space, with a lightweight evaluation function scoring each candidate; chunks predicted as NC (not depicted) are left for manual resolution since this dissertation does not address code generation.

sizes confirm that search over a well-pruned candidate space is computationally feasible. *RQ2 (Partial order)*: 98.6% of resolutions preserve the relative order in which lines appear within V1 and within V2, i.e., lines from V1 are not reordered among themselves, and likewise for V2. *RQ3 (Interleaving)*: 77.4% of resolutions interleave V1 and V2 blocks at most once (a single transition from one version's lines to the other's). *RQ4 and RQ5 (Language constructs)*: these two questions jointly examine how the type of Java construct present in the conflicting chunk shapes the resolution pattern; Campos Junior et al. [4] found this influence to be statistically significant, providing additional feature signal for predictive models.

Search-space reduction and constraint trade-offs. Applying the partial-order constraint alone (RQ2) reduces the search space by **89.2%** for the median-sized conflict, while covering **98.6%** of the dataset; a search algorithm enforcing this constraint would actually fail on only 0.88% of chunks, because roughly a third of the 1.4% partial-order violators are tolerant to reordering (e.g., resolutions consisting only of import statements, where line order is irrelevant) [4]. Combining both constraints (partial order and no-interleaving from RQ3) achieves a stronger reduction of **94.7%**, but narrows coverage to 72.6% of the dataset, excluding 27.4% of real combination-based resolutions. This trade-off motivates retaining only the partial-order constraint in SBCR: the 89.2% search-space reduction is substantial, and the near-complete conflict coverage (98.6%) ensures the tool remains applicable to the vast majority of combination-based cases. In the running example (Figure 1), the CB resolution [V1-L1, V2-L2] satisfies the partial-order constraint trivially: each parent contributes a single line, so no within-parent ordering can be violated. This *Feasibility* step (Figure 2) prunes the search space before the RRHC algorithm begins.

5 A Lightweight Evaluation Function

A search-based method that evaluates each candidate by compiling the merged file and running its test suite faces two fundamental obstacles. First, it would be impractically slow: even modest conflicts may generate many candidates per run, making compilation plus testing a prohibitive oracle. Second, and more fundamentally, tests covering the specific conflicting region may simply not exist, making test execution inapplicable as an oracle, regardless of its speed. Although automated test generation could in principle fill

this gap, the generative AI techniques capable of producing such tests were nascent at the time of this research and too unreliable for systematic use as a fitness oracle. This contribution [5] therefore investigates whether a *parent-similarity* metric can serve as a cheap proxy for resolution quality.

Hypothesis. We hypothesize that, if developer resolutions are structurally similar to the conflicting parents, then a candidate highly similar to both V1 and V2 is likely close to the ground-truth resolution.

Dataset and method. We analyzed 9,998 conflict chunks drawn from 1,062 Java open-source projects. We operationalize textual similarity using Gestalt pattern matching [23], a metric based on the longest common subsequence (LCS), and aggregate the two parent-similarity scores using four candidate functions: minimum, maximum, arithmetic mean, and harmonic mean. For each chunk we generated a set of random candidates and computed: (i) the Gestalt similarity between the actual developer resolution and each parent, and (ii) the Spearman correlation between the candidates' parent-similarity scores and their similarity to the actual resolution, yielding one correlation value per chunk. The median of these per-chunk correlations is then used to compare the four aggregation functions.

Results. Developer resolutions are on average **70%** similar to each parent. Among the four aggregation functions, the arithmetic mean yields the strongest correlation: median $\rho = \mathbf{0.791}$ across conflict chunks, between the mean parent-similarity of a candidate and its similarity to the actual resolution. This strong correlation validates the hypothesis: a candidate that resembles both parents tends to resemble the resolution a developer would produce. For the running example, the CB resolution [V1-L1, V2-L2] shares the `addChild` call with V1 and the extended `getInfobox` call with V2, yielding high Gestalt similarity to both parents and therefore a high mean score, the signal the evaluation function uses to identify it as near-optimal. The similarity function serves as the fitness oracle in the optimization loop shown in Figure 2 and detailed in the next section.

6 SBCR: Search-Based Conflict Resolution

SBCR (*Search-Based Conflict Resolution*) is the final integrative artifact of this dissertation [5]. It operationalizes the feasibility findings (Section 4) and the evaluation function (Section 5) into a fully automated, language-agnostic tool for resolving merge conflicts, occupying the bottom half of the pipeline shown in Figure 2.

Approach. SBCR [5] represents a conflict chunk as an ordered sequence of lines drawn from the two conflicting versions (V1 and V2). The search space is all candidate sequences that can be formed from any subset of these lines, constrained by the partial-order property established in Section 4. Three manipulation operators explore this space: *insert* (add a line from V1 or V2 absent from the current candidate), *remove* (delete a line from the candidate), and *swap* (exchange two adjacent lines). Candidate quality is measured by the evaluation function from Section 5: the arithmetic mean of the Gestalt similarity between the candidate and V1, and between the candidate and V2. The optimization algorithm is *Random Restart Hill Climbing* (RRHC) [24], which repeatedly initializes a random candidate and performs greedy local search until no improving neighbor exists, retaining the best solution found across restarts. Because it operates on plain text lines and requires no parser or training corpus, SBCR is fully language-agnostic. In the running example, RRHC might initialize with V1 verbatim [L1, L2] and reach the ground-truth CB resolution by applying a *remove* (drop V1-L2) followed by an *insert* (add V2-L2), guided at each step by the improvement in mean Gestalt similarity to both parents.

Empirical evaluation. To assess SBCR's performance on real conflicts, the TOSEM study [5] evaluated it on a Java dataset of CB conflict chunks. The results showed a median resolution similarity of **86.5%**, exact matches in **25.2%** of cases, and a median execution time of **2.56 seconds** per conflict. A subsequent empirical study [11] compared SBCR with MergeGen [15], a transformer-based approach that represents the state of the art in learning-based conflict resolution. The evaluation used two datasets comprising projects written in Java, C#, JavaScript, and TypeScript, totaling more than 50,000 conflict chunks. Resolution quality was consistently assessed using Gestalt similarity with respect to the ground-truth developer resolutions.

Results. The comparison reveals complementary strengths between SBCR and MergeGen. MergeGen achieves higher median similarity overall; SBCR's relative performance is strongest on balanced conflicts (where neither parent dominates, so the parent-similarity evaluation function is most discriminating), on exceptionally large conflicts, and on codebases with non-English content, though MergeGen retains higher overall similarity even in these regimes; Section 7 unpacks each regime in detail. The two techniques therefore target different conflict regimes, motivating the hybrid resolution strategies we discuss next.

7 Discussion and Implications

Cumulative coverage. The four contributions form a progressive pipeline. Combining MESTRE's per-strategy accuracy with the prevalence of each direct strategy in the dataset, MESTRE correctly auto-resolves **70.5%** of conflict chunks via direct predictions (V1, V2, CC12, CC21). SBCR addresses the CB subset, which MESTRE intentionally routes to the search-based path. Chunks predicted as NC are left untouched, with their conflict markers preserved for the developer to resolve manually, since this dissertation does not address code-generation strategies. Together, MESTRE and SBCR cover the dominant portion of the workload (V1, V2, CC, and CB), leaving only NC chunks (a minority) for manual resolution. Figure 2 summarizes this coverage story.

Developer workflow integration. Consider a developer invoking `git merge feature-branch`. When conflicts arise, Git marks each conflicting region with <<<<<<<, =======, and >>>>>>> markers, and the developer must manually resolve each chunk before the merge can be committed. This interrupts the development flow and, on large branches, can occupy significant time [21]. With the dissertation's pipeline integrated as an IDE plugin or a pre-commit continuous-integration (CI) step, the workflow changes substantially. Each conflict chunk is automatically extracted and passed to MESTRE. Chunks predicted as a direct strategy (V1, V2, CC12, CC21) are resolved immediately without any developer interaction; this path yields a correct resolution for **70.5%** of all conflicts overall, with the residual error discussed under Risks below. For chunks

predicted as CB, SBCR performs a local search (median 2.56 seconds) and presents the best candidate for lightweight review; the developer's task is reduced from constructing the resolution from scratch to reviewing, and if necessary, adjusting, a pre-filled proposal. Chunks predicted as NC are left with their conflict markers in place, and the developer resolves them by writing new code, as in the traditional workflow. Because SBCR operates on raw conflict markers with no language grammar or trained model, deployment requires no project-specific configuration beyond installing the tool; MESTRE, however, must be trained per-project and therefore benefits from an established merge history.

Conflict regimes and complementarity. The empirical comparison with MergeGen (Section 6) shows that the two tools excel in different regimes. MergeGen achieves higher overall median similarity, driven by performance on *imbalanced* conflicts, where V1 and V2 differ substantially in size and the dominant side provides a larger context for the generation model. SBCR, by contrast, shows its strongest *relative* performance on *balanced* conflicts, where neither parent dominates and the parent-similarity evaluation function (Section 5) is most discriminating, although MergeGen retains higher overall median similarity even in this regime. SBCR also remains robust on exceptionally large conflicts and on codebases containing non-English identifiers, comments, or textual content. MergeGen, being transformer-based, may suffer from context-window limits on large conflicts and from training-corpus biases on non-English content. These complementary failure modes suggest that a hybrid dispatcher, which routes balanced conflicts to SBCR and imbalanced ones to a generative model, would outperform either tool alone. However, such an approach would require GPU infrastructure for the MergeGen component. This direction is further explored by Campos Junior and Murta [11].

Gains and trade-offs relative to learning-based tools. The primary practical gain is a reduction in manual effort: over two-thirds of conflicts are resolved automatically by MESTRE, and chunks predicted as combination-based (CB) arrive pre-filled with a high-similarity candidate from SBCR. Unlike MergeGen [15] and DeepMerge [14], SBCR requires no GPU and no training corpus; it takes only the two conflicting versions as input and runs on commodity hardware, making it suitable for local developer workstations and lightweight CI pipelines. Its language-agnostic design also serves projects in languages where training data is scarce, a known bottleneck for learning-based tools. MergeGen, by contrast, requires fine-tuning on language-specific datasets and a GPU inference machine, and is sensitive to input truncation when conflicts are long.

Risks and limitations. MESTRE's 80.8% accuracy means that roughly one in five predictions is incorrect; an erroneous direct resolution applied silently is harder to catch than an unresolved conflict. We therefore recommend that CI integrations compile and run smoke tests after automatic resolution before committing, rather than trusting the prediction blindly. For new or newly cloned repositories, per-project MESTRE models cannot be trained until sufficient merge history is available (cold-start). In these cases, a cross-project or language-level model can serve as a fallback at reduced accuracy.

Implications for practitioners. Three concrete actions follow from the pipeline. First, deploy MESTRE and SBCR as an IDE plugin or pre-commit CI hook, using the open-source artifacts (Section 10); the commodity-hardware footprint avoids the GPU and training-data requirements of learning-based tools. Second, gate automatic resolutions behind a compile or smoke test before committing, to catch the residual MESTRE error noted under Risks above. Third, where GPU inference is feasible, consider a hybrid dispatcher that routes balanced conflicts to SBCR and imbalanced ones to a generative model [11]; this offers a path to higher overall quality without additional labeling or training effort.

Implications for researchers. The use of parent similarity as a fitness proxy demonstrates that lightweight structural metrics can replace expensive oracles (compilation, test execution) in search-based settings, a finding that may be relevant to other code-manipulation problems such as automated patch generation or code transplantation. The empirical characterization of CB resolutions as partial-order-preserving subsequences provides a reusable methodological template: analogous structural analyses could reduce the search space in line-level manipulation tasks beyond merge conflict resolution. The finding that MESTRE performs best with per-project training reveals that conflict resolution strategies are context-dependent; cross-project generalization remains an open research challenge. Finally, the complementarity between SBCR and MergeGen motivates new research directions: hybrid dispatching, conflict-regime detection, and ensemble approaches that combine the strengths of search-based and generative paradigms.

8 Threats to Validity

Conclusion validity. The evaluation function study (Section 5) reports Spearman $\rho = 0.791$ computed over 9,998 conflict chunks, a sample size that provides high statistical power and stable rank-correlation estimates. Spearman (rather than Pearson) correlation is appropriate given the non-normal distribution of candidate similarity scores. MESTRE's accuracy is estimated via 10-fold stratified cross-validation on 20 projects, which reduces the risk of the reported 80.8% figure being an artifact of a single favorable partition. SBCR's median resolution similarity of 86.5% is derived from a Java evaluation dataset [5]; the broader multilingual comparison spanning more than 50,000 conflict chunks [11] further supports the central-tendency estimate, though variance in the tail (worst-case and imbalanced conflicts) warrants further per-project investigation.

Internal validity. Dataset construction varied across the three studies. MESTRE's dataset (Section 3) applied inclusion criteria such as minimum project age and minimum number of merge commits with recorded resolutions, which may underrepresent projects that rarely merge or that have trivially resolved conflicts, and thereby inflate accuracy estimates. The feasibility (Section 4) and evaluation function (Section 5) studies used broader GitHub mining without such filters; while this reduces selection bias, projects with very few relevant resolutions may contribute noisy signal. Additionally, temporal ordering of commits was not enforced during MESTRE's cross-validation, which could allow features derived from later commits (e.g., commit-message keyword frequency) to influence predictions in a way that would not hold in a real prospective deployment. In all studies, the developer's historical resolution is used as ground truth, implicitly assuming that the recorded resolution

was intentional and representative of the developer's intent. Trivial resolutions (e.g., accepting one side without inspection) could introduce label noise.

Construct validity. Resolution quality is operationalized as Gestalt pattern matching similarity to the developer's ground-truth resolution. This metric captures textual proximity rather than semantic correctness: a syntactically similar but semantically wrong resolution would score highly, while a semantically equivalent but differently phrased resolution would be penalized. Complementary evaluation via compilation and test execution, or developer user studies, would strengthen the conclusions. The taxonomy inherited from Ghiotto et al. [17] (with the CC split adopted by MESTRE) covers the observed resolution space in open-source Java, but its completeness for other languages or domains has not been independently validated. MESTRE's 34 features include Java-specific constructs (import statements, method invocations) whose relevance may not transfer to other languages, limiting the construct validity of the feature design outside the Java ecosystem.

External validity. MESTRE and the feasibility study were evaluated exclusively on open-source Java projects hosted on GitHub; results may not generalize to proprietary codebases, organizations with different branching strategies, or teams of substantially different sizes. SBCR was evaluated on a multilingual dataset (Java, C#, JavaScript, TypeScript), providing initial evidence of cross-language applicability; however, projects in this dataset may overlap with MergeGen's pre-training corpus, so performance on truly unseen ecosystems remains unknown. All studies treat conflict chunks in isolation; in practice, multiple chunks within a single merge commit may have semantic interdependencies that a per-chunk approach does not model, and resolving them independently could produce an inconsistent overall merge result.

9 Related Work

Merge approaches. The standard *line-based* (textual) merge computes line-level differences and flags as a conflict any region modified by both branches [20]. Although universally applicable, it produces *spurious conflicts*: regions flagged as conflicting even though the underlying semantic changes are compatible. Apel et al. [3] and Apel et al. [2] introduced structured and semistructured merge, which exploit language parse trees to match corresponding syntactic elements before applying textual merge within each element. This reduces spurious conflicts but requires language-specific grammars and incurs performance overhead. Cavalcanti et al. [12] further evaluated semistructured merge on a large Java corpus, confirming precision gains at the cost of scalability. Accioly et al. [1] complemented this line by empirically characterizing the conflict patterns that persist after semistructured merge. More recently, Cavalcanti et al. [13] extended the approach with language-specific syntactic separators, further reducing spurious conflicts. These approaches operate *during* the merge and reduce the number of conflicts that arise; they are therefore complementary to, rather than in competition with, the conflict resolution tools surveyed below.

Conflict resolution approaches. Once a textual conflict exists, a separate class of tools attempts to resolve it automatically. Dinella et al. [14] introduced DeepMerge, a sequence-to-sequence neural network trained on conflict–resolution pairs. Svyatkovskiy et al. [25] fine-tune transformer models on real merge histories. Zhang et al. [26] use pre-trained language models to handle both textual and semantic conflicts. Dong et al. [15] introduced MergeGen, which frames resolution as a generation task and currently represents the state of the art. All these approaches require substantial training data and GPU infrastructure, and their performance degrades on conflict patterns underrepresented in the training set. The present dissertation occupies a complementary niche: it relies on lightweight classification (MESTRE) plus search (SBCR) rather than end-to-end learned generation, yielding tools that need only modest per-project training data and run on commodity hardware.

Search-based software engineering. Harman and Jones [18] established SBSE as a paradigm for applying metaheuristic optimization to software engineering problems. SBSE has been applied broadly to problems such as test generation, refactoring, and bug fixing, but had not been systematically applied to conflict resolution before this dissertation. The contribution is not merely the application of SBSE to a new domain, but the accompanying empirical analysis that makes SBSE tractable here: the feasibility study that characterizes the search space, and the evaluation function study that provides a lightweight, reliable fitness signal.

10 Conclusion and Future Work

This paper presented a doctoral dissertation on search-based merge conflict resolution, with four integrated contributions summarized as follows. MESTRE, a per-project ML classifier, predicts resolution strategies with 80.8% accuracy and correctly resolves 70.5% of conflicts automatically. An empirical feasibility study established that 98.6% of CB resolutions respect the partial order of conflicting lines; enforcing this alone reduces the search space by 89.2% while preserving near-complete coverage. Adding a no-interleaving constraint reaches 94.7% reduction, at the cost of excluding 27.4% of resolutions; SBCR therefore retains only the partial-order constraint to maximize coverage. A parent-similarity evaluation function based on Gestalt pattern matching achieved $\rho = 0.791$ correlation with resolution quality, providing an efficient fitness signal for search. Finally, SBCR achieved 86.5% median similarity, exact matches in 25.2% of cases, and a median execution time under 3 seconds [5]; it was shown to be complementary to MergeGen across more than 50,000 conflict chunks [11].

Several directions remain open. Extending MESTRE to multilingual projects would broaden direct coverage. Refining the evaluation function to handle imbalanced conflicts could reduce the performance gap with MergeGen in that regime. Exploring alternative metaheuristics may improve solution quality for larger chunks. Integrating MESTRE and SBCR into developer tooling as an IDE-level assistant, with a hybrid dispatcher routing conflicts to the most appropriate resolver, is the natural next step toward production-grade automated merge conflict resolution.

Artifact availability

The source code, scripts, and supplementary material for SBCR are publicly available [10]. The datasets and replication artifacts for each published contribution are available at the following DOIs: SBES 2022 dataset [7]; extended SBES 2022 dataset [9]; JSERD 2024

artifacts [8]. The empirical comparison between SBCR and MergeGen is reported as a preprint [11]; its replication results[2] and datasets[3] are available at Figshare.

Acknowledgements

This work was partially supported by CAPES (doctoral scholarship), FAPERJ (E26/200.591/2021 and E-26/204.145/2024), CNPq (309300/2023-1 and 155576/2025-9), and NSF. The work was also supported by INES.IA (National Institute of Science and Technology for Software Engineering Based on and for Artificial Intelligence) (https://www.ines.org.br), CNPq grant 408817/2024-0. Finally, the authors thank Universidade Federal Fluminense (UFF) for institutional support. Claude (Anthropic) was used for manuscript editing.

References

[1] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. 2018. Understanding Semi-Structured Merge Conflict Characteristics in Open-Source Java Projects. *Empirical Software Engineering* 23, 4 (2018), 2051–2085. doi:10.1007/s10664-017-9586-1

[2] Sven Apel, Olaf Leßenich, and Christian Lengauer. 2012. Structured Merge with Auto-Tuning: Balancing Precision and Performance. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*. Association for Computing Machinery, New York, NY, USA, 120–129.

[3] Sven Apel, Jörg Liebig, Benjamin Brandl, Christian Lengauer, and Christian Kästner. 2011. Semistructured Merge: Rethinking Merge in Revision Control Systems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, Szeged, Hungary, 190–200. doi:10.1145/2025113.2025141

[4] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2024. How Code Composition Strategies Affect Merge Conflict Resolution? *Journal of Software Engineering Research and Development* 12, 1 (2024), 13:1–13:24. doi:10.5753/jserd.2024.3638

[5] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2026. Towards a Feasible Evaluation Function for Search-Based Merge Conflict Resolution. *ACM Transactions on Software Engineering and Methodology* 35, 5, Article 131 (April 2026), 35 pages. doi:10.1145/3748256

[6] Heleno de S. Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2022. Towards Merge Conflict Resolution by Combining Existing Lines of Code. In *Proceedings of the 36th Brazilian Symposium on Software Engineering (SBES '22)*. ACM, Maringá, Brazil, 425–434. doi:10.1145/3555228.3555229 Distinguished Paper Award.

[7] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2022. Artifacts for the SBES 2022 Submission on the Analysis of Merge Conflicts Resolved with Combination of the Conflicting Lines. doi:10.6084/m9.figshare.19705600

[8] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2023. Artifacts for the JSERD Submission: How Code Composition Strategies Affect Merge Conflict Resolution? doi:10.6084/m9.figshare.24012579

[9] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2024. Merge Conflicts Dataset Used in SBES 2022 Paper. doi:10.6084/m9.figshare.26831818

[10] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2025. SBCR: Search-Based Conflict Resolution – Source Code and Supplementary Material. https://github.com/gems-uff/sbcr.

[11] Heleno de Souza Campos Junior and Leonardo Gresta Paulino Murta. 2026. LLM-based vs. Search-based Merge Conflict Resolution: An Empirical Study of Competing Paradigms. https://arxiv.org/abs/2605.16646. arXiv:2605.16646 [cs.SE]

[12] Guilherme Cavalcanti, Paulo Borba, and Paola Accioly. 2017. Evaluating and Improving Semistructured Merge. *Proceedings of the ACM on Programming Languages (OOPSLA)* 1 (2017), 59:1–59:27.

[13] Guilherme Cavalcanti, Paulo Borba, Leonardo dos Anjos, and Jonatas Clementino. 2024. Semistructured Merge with Language-Specific Syntactic Separators. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*. IEEE, 1032–1043.

[14] Elizabeth Dinella, Todd Mytkowicz, Alexey Svyatkovskiy, Christian Bird, Mayur Naik, and Shuvendu Lahiri. 2023. DeepMerge: Learning to Merge Programs. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1599–1614. doi:10.1109/TSE.2022.3183955

[15] Jinhao Dong, Qihao Zhu, Zeyu Sun, Yiling Lou, and Dan Hao. 2023. Merge Conflict Resolution: Classification or Generation?. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE '23)*. IEEE, Luxembourg, Luxembourg, 1652–1664. doi:10.1109/ASE56229.2023.00155

[16] Paulo Elias, Heleno de S. Campos Junior, Eduardo Ogasawara, and Leonardo Gresta Paulino Murta. 2023. Towards Accurate Recommendations of Merge Conflicts Resolution Strategies. *Information and Software Technology* 164 (2023), 107332. doi:10.1016/j.infsof.2023.107332

[17] Gleiph Ghiotto, Leonardo Murta, Márcio Barros, and André van der Hoek. 2020. On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by GitHub. *IEEE Transactions on Software Engineering* 46, 8 (2020), 892–915. doi:10.1109/TSE.2018.2871083

[18] Mark Harman and Bryan F. Jones. 2001. Search-Based Software Engineering. *Information and Software Technology* 43, 14 (2001), 833–839. doi:10.1016/S0950-5849(01)00189-6

[19] Bakhtiar Khan Kasi and Anita Sarma. 2013. Cassandra: Proactive Conflict Minimization through Optimized Task Scheduling. In *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*. IEEE, San Francisco, CA, USA, 732–741. doi:10.1109/ICSE.2013.6606619

[20] Tom Mens. 2002. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering* 28, 5 (2002), 449–462. doi:10.1109/TSE.2002.1000449

[21] Nicholas Nelson, Caius Brindescu, Shane McKee, Anita Sarma, and Danny Dig. 2019. The Life-Cycle of Merge Conflicts: Processes, Barriers, and Strategies. *Empirical Software Engineering* 24, 5 (2019), 2863–2906. doi:10.1007/s10664-018-9674-x

[22] Shaun Phillips, Jonathan Sillito, and Rob Walker. 2011. Branching and Merging: An Investigation into Current Version Control Practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '11)*. Association for Computing Machinery, New York, NY, USA, 9–15.

[23] John W Ratcliff, David E Metzener, et al. 1988. Pattern matching: The gestalt approach. *Dr. Dobb's Journal* 13, 7 (1988), 46.

[24] Stuart Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach* (4 ed.). Pearson.

[25] Alexey Svyatkovskiy, Sarah Fakhoury, Negar Ghorbani, Todd Mytkowicz, Elizabeth Dinella, Christian Bird, Jinu Jang, Neel Sundaresan, and Shuvendu K. Lahiri. 2022. Program Merge Conflict Resolution via Neural Transformers. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*. ACM, Singapore, Singapore, 822–833. doi:10.1145/3540250.3549163

[26] Jialu Zhang, Todd Mytkowicz, Mike Kaufman, Ruzica Piskac, and Shuvendu K. Lahiri. 2022. Using Pre-trained Language Models to Resolve Textual and Semantic Merge Conflicts. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22)*. Association for Computing Machinery, New York, NY, USA, 77–88.

[2] https://figshare.com/s/b3cdd351d077a9b08121

[3] https://figshare.com/s/d196f4ccb3ef34d2e770