Robust Lazy Conflict Detection via Multi-Conflict Extraction and Genetic Diversity Control

Viet-Man Le, Lukas André Feldgrill, Alexander Felfernig

Graz University of Technology, Graz, Austria
v.m.le@tugraz.at, feldgrill@student.tugraz.at, alexander.felfernig@tugraz.at

Abstract

Detecting minimal conflict sets is essential for providing meaningful feedback in knowledge-based configuration. While *lazy conflict detection* addresses runtime efficiency by predetermining conflict sets offline using a genetic algorithm, it suffers from low conflict coverage, stagnation, and instability. We propose a robust enhancement that integrates *multi-conflict extraction* and *genetic diversity control* to overcome these limitations. Our method extends conflict discovery per evaluation and introduces three diversity mechanisms: full population reproduction, weighted genetic operators, and adaptive extinction. Empirical evaluations on five real-world configuration knowledge bases show that our approach recovers up to 85% of conflict sets, reduces solver calls by up to 73%, and achieves higher result stability. These improvements demonstrate the scalability and reliability of enhanced lazy conflict detection for interactive configuration systems.

Artifact — https://github.com/AIG-ist-tugraz/robust-lcd
Appendix — https://doi.org/10.5281/zenodo.17610291

Introduction

Detecting minimal conflict sets (MCSs), also known as minimal unsatisfiable subsets (MUSs), is a central task in knowledge-based configuration (Mittal and Frayman 1989; Fleischanderl et al. 1998; Felfernig et al. 2014). When user requirements lead to inconsistency, identifying a minimal explanation is essential for interactive systems to provide meaningful and actionable feedback (Nielsen 1994; Junker 2006). Existing approaches, such as QUICKXPLAIN (Junker 2004), efficiently identify one conflict set per run but remain impractical for large-scale configuration knowledge bases (CKBs) due to their high solver overhead (O'Sullivan et al. 2007; Jannach, Schmitz, and Shchekotykhin 2015; Vidal et al. 2021; Le et al. 2023).

To address this scalability issue, *lazy conflict detection* (LCD) (Uran and Felfernig 2018) proposes an offline strategy: minimal conflict sets are predetermined using a genetic algorithm and stored in a lookup table for real-time retrieval during configuration. While this significantly reduces solver calls at runtime (Le et al. 2024), the approach suffers from several critical limitations: (1) **coverage gaps**, where only 20–30% of MCSs are discovered; (2) **stagnation**, where the evolutionary process prematurely converges, limiting further exploration; and (3) **instability**, where repeated executions yield inconsistent outcomes.

These limitations arise from well-known challenges in evolutionary search, such as limited exploration, dependency on the initial population, and unbalanced selection pressure (Mitchell 1998; Katoch, Chauhan, and Kumar 2021). Nevertheless, prior work has not systematically addressed them in the context of conflict detection (Bessiere et al. 2013; Carneiro, Novais, and Neves 2013).

In this page, we address these issues via four complementary enhancements, organized into two strategic directions:

- **Multi-Conflict Extraction**: Enhances conflict discovery by extracting multiple minimal conflict sets from each individual in the population. This is achieved by applying the HSDAG algorithm (Hitting Set Directed Acyclic Graph) (Reiter 1987; Greiner, Smith, and Wilkerson 1989) in combination with QUICKXPLAIN to systematically enumerate MCSs associated with a given configuration.
- **Genetic Diversity Control**, which comprises three mechanisms:
  - *Full population reproduction*, including consistent individuals in crossover;
  - *Weighted genetic operators*, prioritizing parents more likely to yield conflicts;
  - *Adaptive extinction*, that resets the population upon detecting stagnation.

We empirically evaluate these mechanisms, both individually and in combination, on five real-world CKBs ranging from compact models (Arcade) to large-scale benchmarks (EA). Our results show that the proposed method (1) achieves up to **85%** conflict set coverage, significantly outperforming the baseline; (2) reduces solver calls by up to **73%** through more efficient conflict discovery; (3) improves result stability, with variance reductions of up to **50%** over multiple independent runs; and (4) generalizes effectively across CKBs of varying size and constraint complexity.

Overall, this work demonstrates that combining multi-conflict extraction with diversity-aware genetic search yields a scalable and robust framework for offline conflict predetermination in configuration systems.

Background and Problem Statement

Knowledge-based configuration tasks are commonly modeled as *constraint satisfaction problems* (CSPs) (Rossi, van Beek, and Walsh 2006), where the objective is to find a valid assignment to a set of variables subject to domain constraints and user-defined requirements.

Configuration as a CSP

A *configuration task* and its *configuration* (solution) are defined as follows:

Definition 1 (Configuration task and Configuration knowledge base)**.** A configuration task can be defined as a CSP (V, D, C) where (1) $V = \{v_1 \,..\, v_n\}$ is a set of finite domain variables, (2) $D = \{dom(v_i) : v_i \in V\}$ is a set of domain definitions for each variable $v_i \in V$, and (3) $C = C_{KB} \cup C_R$ is a set of constraints restricting possible solutions of a configuration task. C_{KB} represents a set of domain-specific constraints, and C_R represents a set of user requirements. Finally, (V, D, C_{KB}) is denoted as a configuration knowledge base.

Definition 2 (Configuration)**.** A configuration (solution) S for a given configuration task (V, D, C) is an *assignment* $A = \{v_1 = a_1 \,..\, v_n = a_n\}$, $a_i \in dom(v_i)$. S is *valid* if it is *complete* (each variable in V has a value) and *consistent* (S fulfills the constraints in C).

When $C_R \cup C_{KB}$ is inconsistent, no valid configuration exists. In such situations, we are interested in explanations as to why no solution could be identified. MCSs, also known as MUSs, are a means often used to explain such inconsistent situations (O'Sullivan et al. 2007; Bendík and Černá 2020). In the following, we formally define *minimal conflict sets* and discuss how to identify them to support the resolution of inconsistent configuration tasks.

Conflict Sets and Diagnoses

Conflict sets are constraint sets that are responsible for an inconsistency, i.e., a situation in which no solution can be found. We use *consistent*(C) to indicate a consistent constraint set C, and *inconsistent*(C) for situations without a solution for C.

Definition 3 (Conflict set)**.** A *conflict set* is a set $CS \subseteq C_R$: *inconsistent*($CS \cup C_{KB}$). CS is *minimal* iff $\nexists CS' \subset CS$: *inconsistent*($CS' \cup C_{KB}$).

Definition 3 introduces the concept of conflict set minimality on the basis of *subset minimality* (i.e., not minimal cardinality). This means if CS is a minimal conflict, no proper subset of CS is a minimal conflict. With conflict set minimality, each conflict can be resolved by removing one element from the corresponding conflict set.

One of the most efficient and widely adopted algorithms to conflict detection is QUICKXPLAIN (Rodler 2022). This algorithm was introduced by Junker (Junker 2004), which supports the determination of a minimal conflict set in a given set of constraints C. Following a divide-and-conquer search regime, the underlying idea of QUICKXPLAIN is to divide the consideration set into two subsets. If the first subset is inconsistent, then conflict detection should be applied to this subset. The second subset must not be further analyzed since at least one conflict exists in the first subset. This way, the consideration set can be reduced by half with a single consistency check.

QUICKXPLAIN determines exactly one conflict set per computation. In order to identify all minimal conflicts, this algorithm has to be combined with the HSDAG-based diagnosis algorithm (Reiter 1987; Greiner, Smith, and Wilkerson 1989). HSDAG repeatedly activates a conflict detection algorithm (e.g., QUICKXPLAIN) and analyzes different possibilities to resolve the returned conflict. This helps to identify all minimal conflict sets and corresponding minimal diagnoses in the consideration set C (Felfernig, Schubert, and Zehentner 2012; Nica et al. 2013). For further details of the algorithm, we refer to (Reiter 1987) and (Greiner, Smith, and Wilkerson 1989).

To resolve all conflicts, we need to identify corresponding hitting sets, also known as diagnoses (Reiter 1987), that have to be adapted or deleted to make the user requirements consistent with the knowledge base. Based on the definition of a conflict set, we now introduce the definition of a *diagnosis task* and a corresponding *diagnosis*.

Definition 4 (Diagnosis task)**.** A diagnosis task for a configuration task (V, D, C) can be defined by a tuple (C_R, C_{KB}), where C_R is a set of user requirements to be analyzed and C_{KB} is a set of constraints specifying the configuration knowledge base assumed to be consistent.

Definition 5 (Diagnosis)**.** A *diagnosis* Δ of a diagnosis task (C_R, C_{KB}) is a set $\Delta \subseteq C_R$: *consistent*($C_R \setminus \Delta \cup C_{KB}$). Δ is *minimal* iff $\nexists \Delta' \subset \Delta$: *consistent*($C_R \setminus \Delta' \cup C_{KB}$).

Lazy Conflict Detection

Lazy conflict detection is an offline strategy designed to reduce the number of solver calls during runtime in knowledge-based configuration. Instead of identifying minimal conflict sets on demand, the algorithm precomputes a set of conflicts by simulating user requirement scenarios using a genetic algorithm (GA). Each individual in the GA represents a set of user requirements. If an individual is inconsistent with the CKB, a conflict set is extracted using QUICKXPLAIN. Discovered conflicts are stored in a lookup table for real-time reuse during interactive sessions (Le et al. 2024).

This precomputation strategy significantly improves runtime responsiveness and enables scalable conflict detection (Le et al. 2024). However, the baseline method faces several limitations, including low coverage of MCSs, population stagnation, and inconsistent results across runs. These weaknesses primarily stem from premature convergence in the GA and the limited conflict information extracted per iteration. We address these limitations in the following sections by introducing enhancements that improve both the effectiveness and stability of lazy conflict detection.

Lazy Conflict Detection: Baseline Approach

LCD, originally introduced by Uran and Felfernig (2018), enables precomputation of conflict sets using a genetic algorithm, thereby reducing the runtime cost of online conflict detection. In this section, we present the baseline algorithm and highlight its core limitations.

Algorithm 1: Lazy Conflict Detection (Baseline)

```
1: Population ← InitializePopulation(N, P_∅)
2: ConflictBase ← ∅
3: while time limit not reached do
4:    Parents ← ∅
5:    for each C_R ∈ Population do
6:       if inconsistent(C_KB ∪ C_R) then
7:          CS ← QuickXPlain(C_R, C_KB)
8:          ConflictBase ← ConflictBase ∪ {CS}
9:          Parents ← Parents ∪ Resolve(C_R, CS)
10:      end if
11:   end for
12:   NewGen ← Crossover(Parents)
13:   Population ← Mutate(NewGen, P_m)
14: end while
15: return ConflictBase
```

Overview of the Baseline Algorithm

The baseline approach uses a GA to explore sets of user requirements. Each individual represents a set of user requirements C_R. The overall process (see *Algorithm 1*) consists of repeated evaluation and evolution steps until a time or iteration budget is reached.

Initialization. The algorithm begins by generating an initial population of N individuals. Each individual is formed by randomly assigning values to user requirements, with a probability $P_\emptyset$ that a requirement remains unassigned (*line 1*). This encourages diversity in the starting population, mixing consistent and inconsistent sets of user requirements.

Evaluation. Each individual is evaluated for consistency with the configuration knowledge base C_{KB} (*line 6*). If an individual is found to be inconsistent, the algorithm invokes QUICKXPLAIN to extract a minimal conflict set (*line 7*), which is then stored in the global conflict base (*line 8*). The resulting conflict set is also forwarded to the $Resolve$ function, which generates candidate parents for the next generation (*line 9*).

The $Resolve$ function plays a central role in conflict-driven exploration. Given an inconsistent individual C_R and its associated conflict set $CS \subseteq C_R$, it generates a set of modified individuals by removing one constraint at a time from CS. These variants represent minimal repairs, each corresponding to a potential diagnosis.

Formally, for each $c \in CS$, a new candidate is created as $C'_R = C_R \setminus \{c\}$. All such resolved variants, regardless of whether they are consistent, are added to the parent pool. Notably, the original inconsistent individual is excluded from reproduction. This strategy ensures that genetic material is shaped by minimal adaptations of known conflicts, encouraging diversity without redundant rediscovery of previously analyzed inconsistencies.

Reproduction. The reproduction step generates a new population by applying the genetic operators, crossover and mutation, to the parent pool constructed during the evaluation phase (*lines 12–13*).

Crossover. Pairs of parents are randomly selected from the set of resolved individuals. For each pair, a new offspring is generated by recombining their user requirement assignments. Recombination is performed by iterating over the union of user requirement variables that appear in either parent, ensuring that no relevant assignment is omitted. The value of each variable in the offspring is determined as follows:

- If both parents assign the same value to a variable, the offspring inherits that value.
- If one parent assigns a value and the other leaves it unassigned, the offspring inherits the assigned value with probability 0.5.
- If the parents assign different values, the offspring randomly adopts one of the two.

This crossover strategy preserves structural traits from both parents while introducing variation in the offspring population. The process is repeated until the new population reaches the target size N.

Mutation. After crossover, mutation is applied to each variable in the offspring with probability P_m. If a variable is selected for mutation, it is assigned a new value randomly drawn from its domain, regardless of whether it was previously assigned. This mutation mechanism promotes exploration by introducing new set of user requirements and helps prevent premature convergence.

The baseline algorithm does not incorporate any explicit fitness function or ranking mechanism. As a result, the only form of selection pressure arises from the conflict-driven repair process that determines which individuals are eligible to become parents.

At the end of each iteration, the newly generated and mutated offspring replace the current population for the next generation. The algorithm then repeats this process iteratively until the predefined time limit is reached (*line 3*).

Limitations

While the baseline algorithm offers a scalable alternative to online conflict detection, it exhibits three major limitations that affect its practical effectiveness.

First, the algorithm often suffers from *coverage gaps*, discovering only a small fraction (typically 20–30%) of the minimal conflict sets. This issue stems from the fact that each inconsistent individual yields at most one conflict set via QUICKXPLAIN, which limits the diversity of conflicts that can be extracted in each generation.

Second, the search process is vulnerable to *premature convergence*. Due to the absence of an explicit fitness function or diversity preservation mechanisms, the population tends to converge quickly toward regions of the search space that produce conflicts early, thereby reducing the likelihood of exploring alternative conflict-inducing user requirements. This stagnation often leads to redundant rediscovery of similar conflict sets across generations.

Third, the algorithm exhibits *instability across runs*. Because the GA relies on random initialization and stochastic operators, different executions may yield substantially different sets of conflicts. Without dedicated control over popula-

tion diversity, the outcome is sensitive to initial conditions and may not be repeatable.

These limitations arise from structural characteristics of the baseline GA: single-conflict extraction per individual, absence of explicit diversity preservation, and random recombination without guidance. To overcome these challenges, we introduce several enhancements aimed at increasing conflict coverage, reducing solver calls, and improving stability. These enhancements are presented in the next section.

Enhanced Lazy Conflict Detection

To address the limitations of the baseline, we introduce four enhancements grouped into two strategic directions: Multi-Conflict Extraction and Genetic Diversity Control. These mechanisms are designed to increase conflict coverage, reduce solver effort, and improve robustness across runs.

Multi-Conflict Extraction

The baseline algorithm applies QUICKXPLAIN to extract a single minimal conflict set per inconsistent individual (*line 7* in *Algorithm 1*). This introduces a critical limitation: if an individual contains multiple conflict sets, only one of them is identified per individual. The idea is that other conflict sets will emerge in later generations due to variation introduced by $Resolve$. However, this is not guaranteed. In fact, the genetic operators, crossover and mutate, may remove constraints involved in yet-undetected conflicts, leading to permanent loss of valuable conflict information.

To address this issue, we enhance the evaluation step by integrating QUICKXPLAIN with the HSDAG algorithm (Reiter 1987; Greiner, Smith, and Wilkerson 1989). This integration enables the discovery of multiple minimal conflict sets from a single inconsistent individual. Specifically, HSDAG orchestrates repeated invocations of QUICKXPLAIN to systematically enumerate conflict sets within the given user requirements. This enhancement increases the likelihood of capturing the complete set of conflicts before the user requirements is altered through genetic operators.

The conflict identification process terminates once a predefined limit on the number of extracted conflict sets (e.g., 10) is reached. This cap is essential to balance the benefits of broader conflict discovery with the computational cost of the conflict identification process.

Overall, this enhancement improves the conflict coverage rate, particularly for large-scale or densely constrained knowledge bases, where multiple conflicts per individual are common. By discovering several conflicts in a single evaluation step, the algorithm reduces reliance on stochastic discovery and prevents premature loss of critical conflict sets.

Genetic Diversity Control

To address the issues of premature convergence and instability, we introduce three complementary mechanisms that reshape the genetic search dynamics by promoting diversity and strategically guiding selection.

Full Population Reproduction. The baseline algorithm restricts the parent pool to resolved variants derived from inconsistent individuals. In contrast, this strategy incorporates both consistent individuals and resolved variants into the parent pool. By broadening the diversity of traits available for recombination, this approach enhances exploratory potential and mitigates the risk of stagnation in conflict-prone regions.

Weighted Genetic Operators. To further guide the genetic search while maintaining population diversity, we introduce a set of *weighted genetic operators*. These mechanisms dynamically adjust the likelihood of selection and influence inheritance dynamics, prioritizing individuals that previously contributed to conflict discovery.

Weighted Conflicts. This mechanism biases parent selection toward individuals that contribute further conflict sets. During the evaluation phase of LCD, inconsistent individuals are analyzed to identify multiple minimal conflict sets. Each resulting *resolved variant*, created by removing a diagnosis that resolves all identified conflict sets, is assigned a weight proportional to the number of conflicts detected in the original individual.

Formally, if an individual yields k minimal conflict sets and l corresponding diagnoses, then each of the l resolved variants receives a weight of k, capturing the conflict density of its origin. In contrast, consistent individuals are assigned a base weight of zero. The probability of selecting a resolved variant i for crossover is defined as:

$$P(i) = \frac{1 + w_i}{\sum_j (1 + w_j)}$$

where w_i is the number of conflicts identified in the individual i. This prioritization directs the search toward areas likely to yield additional conflicts while still allowing occasional selection of consistent individuals for exploration.

Conflict Redundancy Avoidance (CRA). Since multiple resolved variants can stem from the same inconsistent individual, weighted selection may inadvertently select closely related parents, reducing genetic diversity. To prevent this, we introduce *Conflict Redundancy Avoidance*.

Each group of resolved individuals derived from a single inconsistent origin is assigned a unique identifier. During crossover, the algorithm ensures that no two selected parents share the same origin ID. This safeguard reduces the chance of producing offspring that replicate known conflicts, thereby enhancing coverage of previously unexplored regions in the search space.

Weighted Crossover. Beyond influencing which parents are selected, we also bias the crossover process based on parental weights. Let $\Delta_w = w_1 - w_2$ denote the weight difference between two selected parents. The probability that the offspring inherits a trait from the higher-weight parent is computed using the following smoothed function:

$$probability = \arctan\left(\frac{\Delta_w}{\pi^c}\right) \cdot 0.3 + 0.5$$

Here, c is a tunable constant that controls the steepness of the curve, and we set $c = 2$ in our implementation. As shown in *Figure 1*, the function yields a probability of 0.5 when both parents have equal weights ($\Delta_w = 0$), ensuring uniform inheritance. As the absolute value of Δ_w increases, the probability gradually shifts toward the higher-weight parent, up

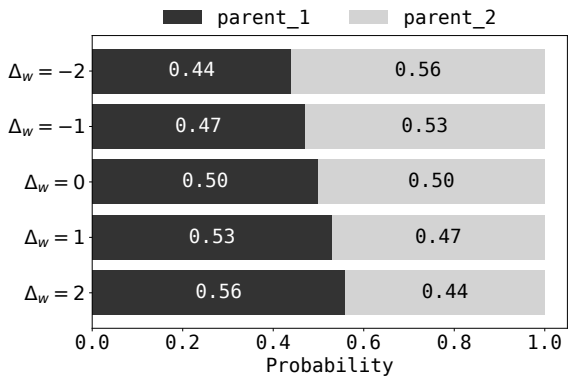

Figure 1: Weighted crossover probability as a function of parent weight difference (Δ_w), with $c = 2$. Positive Δ_w favors parent 1; negative Δ_w favors parent 2.

to a maximum of approximately 0.8. This smooth, bounded adjustment ensures that promising traits are more likely to be passed on without fully excluding the other parent.

This mechanism helps reinforce successful conflict-yielding patterns while avoiding excessive exploitation. Combined with CRA, it supports a balance between intensification and diversification.

Together, these weighted genetic operators, including conflict-based weighting, redundancy-aware pairing, and weight-sensitive recombination, enhance the efficiency, stability, and robustness of the genetic search. They enable the algorithm to concentrate solver effort on promising areas of the user requirement space while maintaining the diversity necessary to uncover new conflicts.

Adaptive Extinction. A well-known issue in genetic algorithms is the tendency toward convergence, especially in aging populations. Over time, genetic diversity diminishes as dominant traits proliferate, reducing the algorithm's ability to explore new regions of the search space. This problem is particularly acute in smaller populations or in domains with sparse conflict distribution, where the search may become trapped in local basins of attraction.

To counteract this stagnation, we implement an *adaptive extinction* mechanism that monitors the algorithm's progress and resets the population when exploration stalls. Specifically, if a predefined number of generations pass without discovering any new minimal conflict sets, the entire population is discarded and replaced with a randomly initialized one. This *soft reset* rejuvenates the search process, allowing the algorithm to escape local optima and re-enter unexplored regions of the user requirement space.

The extinction threshold, i.e., the number of stagnant generations tolerated before extinction is triggered, is configurable and can be tailored to the complexity of the configuration knowledge base. In practice, this threshold is chosen conservatively to avoid premature resets while still responding to genuine stagnation.

Extinction can also be combined with a global termination criterion that halts the algorithm after a fixed number of extinction cycles. When used alongside the *Weighted Conflicts* mechanism, adaptive extinction plays a complementary role: the weighting scheme drives the search toward conflict-rich areas of the search space, which are eventually exhausted, triggering extinction and encouraging exploration of structurally different conflict regions.

By introducing periodic resets conditioned on search stagnation, adaptive extinction preserves long-term diversity, improves coverage of the conflict set space, and prevents overfitting to known conflicts. It is particularly effective in large or heterogeneous CKBs, where conflicts may be scattered across diverse regions of the user requirement space.

CKB	Arcade	FQA	B2C	BusyBox	EA
$\|V\|$	47	101	194	683	1,069
$\|C_{KB}\|$	66	101	233	405	2,670
$\|\mathbb{N}\|$	142	366	267	161	∼8,981

Table 1: Configuration knowledge bases (CKBs) used for our experiments (Arcade=Arcade Game PL, FQA=Feature model for Functional Quality Attributes, B2C=Feature model for a B2C system with fixed priced products, BusyBox=Feature model for the software suite BusyBox, and EA=Feature model for EA 2468). $|\mathbb{N}|$ represents the number of known minimal conflict sets, while the symbol $\sim$ indicates that additional minimal conflict sets may exist.

Empirical Evaluation

We evaluate the effectiveness, efficiency, and stability of our enhancements to *lazy conflict detection* across five real-world configuration knowledge bases (CKBs) collected from the S.P.L.O.T. feature model repository (Mendonca, Branco, and Cowan 2009) and the Diverso Lab's benchmark (Heradio et al. 2022). These models (see *Table 1*) span a range of sizes and constraint densities, from compact domains like Arcade to large-scale industrial models such as EA. Our goal is to assess three key dimensions: conflict set coverage, solver efficiency, and run-to-run stability, with detailed results summarized in *Table 3* and trade-off/convergence plots shown in *Figures 2* and *3*. Complete results, extended analyses, and full plots are provided in Appendix.

To ensure a comprehensive evaluation, we compare a baseline GA with eight enhanced configurations, each incorporating different combinations of four mechanisms (see *Table 2*): *multi-conflict extraction*, *full population reproduction*, *weighted genetic operators*, and *adaptive population extinction*. Each configuration is executed 20 times, and evaluated along three metrics: the average number of discovered conflict sets (*coverage*), the average number of consistency checks per conflict (*efficiency*), and the standard deviation of coverage (*stability*). These metrics are normalized per model using min–max scaling and aggregated into an overall score via the formula: $Overall = 0.4 \times NormCoverage + 0.4 \times NormEfficiency + 0.2 \times NormStability$. This aggregation favors configurations that balance performance and robustness, ensuring that unstable but high-coverage runs do not dominate the ranking.

Configuration	Multi-CS Extraction	Full Population Reproduction	Weighted Conflicts	Conflict Redundancy Avoidance	Weighted Crossover	Adaptive Extinction
Baseline						
Multi-CS	✓					
Full Pop	✓	✓				
Weighted Basic	✓	✓	✓			
Weighted Advanced	✓	✓	✓	✓		
Weighted Full	✓	✓	✓	✓	✓	
Extinction						✓
Ext+Multi-CS	✓					✓
All Features	✓	✓	✓	✓	✓	✓

Table 2: Nine configurations.

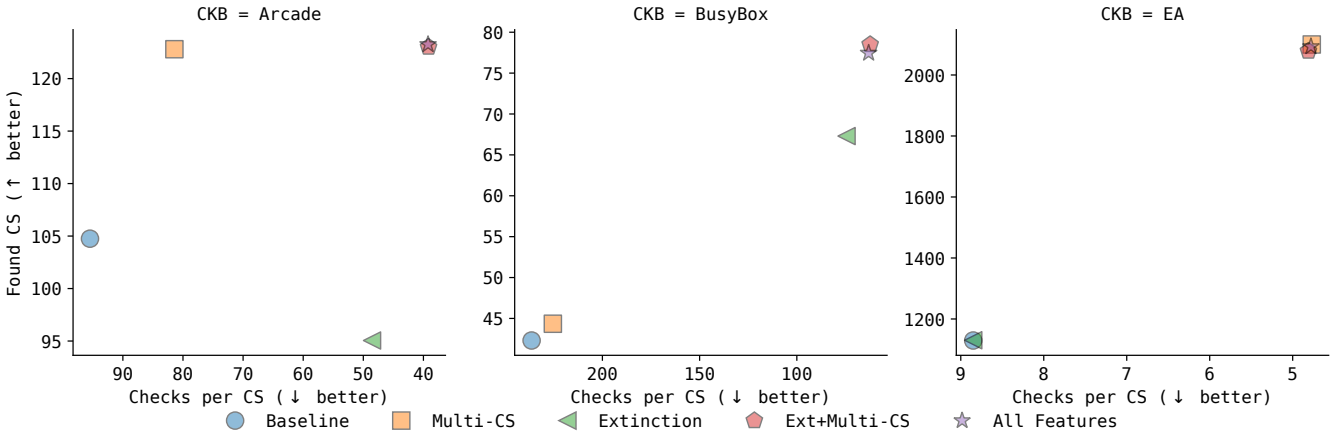

Figure 2: Trade-off between conflict set coverage (Found CS) and solver effort (Checks/CS) for three representative configuration knowledge bases. Configurations closer to the top-right indicate better coverage and lower solver cost. *All Features* and *Ext+Multi-CS* consistently lie near the Pareto frontier, while the *Baseline* lags behind in all settings. Full results for all models are shown in *Figure 1* of the Appendix.

All experiments were conducted on an Apple M1 Pro (8 cores, 16GB RAM) using the CHOCO solver for consistency checks (Prud'homme and Fages 2022). The population size was fixed to $N = 100$, with no assignment rate $P_\emptyset = 0.7$, mutation rate $P_m = 0.1$, and a maximum of 100 generations. For *Multi-CS*, the conflict count was capped at 10 per individual. Extinction was triggered after 3 stagnant generations, with a maximum of 10 extinction cycles. Results across diverse knowledge bases indicate that performance remains stable under fixed parameter settings, suggesting robustness without the need for per-model tuning.

We first examine results on three representative CKBs. On the Arcade model (the leftmost subfigure in *Figure 2*), the *All Features* configuration delivers the strongest overall performance, achieving the highest coverage and a 58.9% reduction in consistency checks relative to the baseline. *Ext+Multi-CS* yields the best solver efficiency (39.15 checks per conflict), while *Full Pop* achieves the lowest variance (*Table 3*). Convergence analysis in *Figure 3a* shows that *All Features* rapidly accumulates conflict sets in early generations, whereas the baseline configuration stagnates early. This highlights the benefit of diversity preservation and adaptive resets in avoiding premature convergence.

In the BusyBox model (the middle subfigure in *Figure 2*), which is highly constrained and solver-intensive, *Ext+Multi-CS* again outperforms all other configurations in both coverage and efficiency, recovering over 85% more conflicts while requiring 73% fewer consistency checks than the baseline. *Extinction* alone already provides significant gains, especially in early stages (*Figure 3b*), confirming its effectiveness in dense problem spaces. The *Weighted Basic* configuration achieves the best stability here (*Table 3*), showing that simpler enhancements may still yield benefits under certain conditions.

The EA model (the rightmost subfigure in *Figure 2*) is the largest and sparsest of the benchmarks. Here, *Multi-CS* emerges as the best-performing configuration across all dimensions, with the highest conflict set coverage (2,096.5 on average) and the lowest solver effort (4.77 checks per conflict). While *All Features* achieves similar results, *Full Pop* stands out as the most stable configuration, earning the best overall score (*Table 3*). Notably, *Extinction* contributes little in this context, as stagnation is not a limiting factor in such

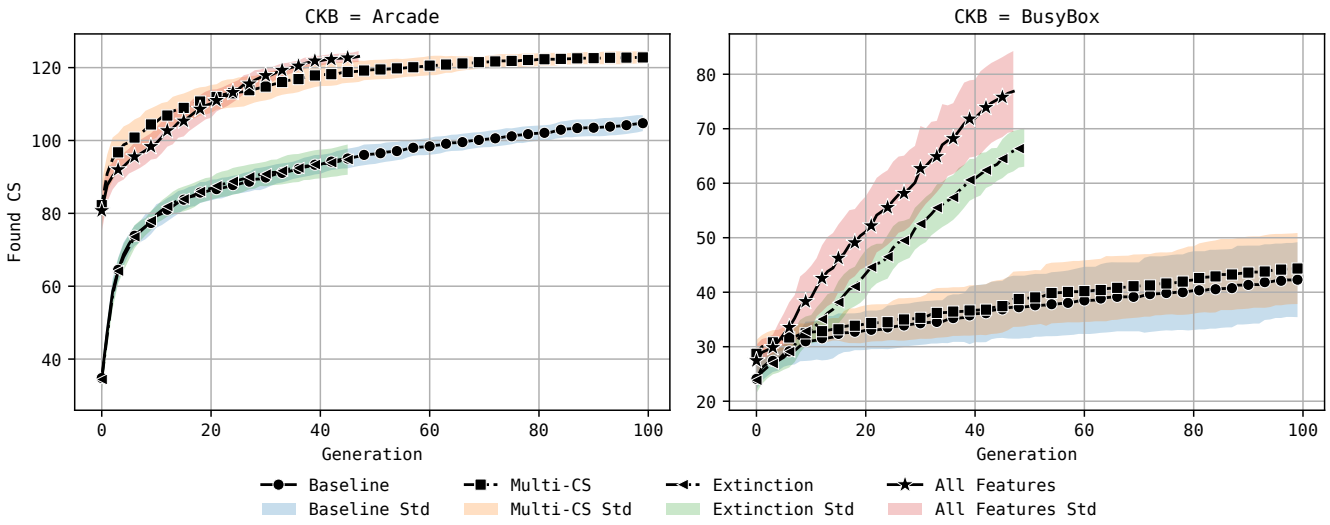

Figure 3: Convergence of conflict set discovery over generations for Arcade and BusyBox models. Solid lines denote the mean number of discovered conflict sets across 20 runs; shaded areas represent standard deviation. *All Features* and *Extinction* achieve faster and more stable convergence, particularly in the early generations. *Baseline* stagnates early, while *Multi-CS* improves total coverage but converges more slowly. Full plots are available in Appendix *Figure 2*.

CKB	Best Coverage	Best Efficiency	Best Stability	Best Overall
Arcade	All Features	Ext+Multi-CS	Full Pop	All Features
FQA	Ext+Multi-CS	All Features	Baseline	All Features
B2C	Ext+Multi-CS	Ext+Multi-CS	Full Pop	All Features
BusyBox	Ext+Multi-CS	Ext+Multi-CS	Weighted Basic	Extinction
EA	Multi-CS	Multi-CS	Extinction	Full Pop

Table 3: Best configurations per model and metric: coverage, efficiency (lower is better), stability (lower std. dev.), and overall score (weighted aggregate score (0.4 coverage, 0.4 efficiency, 0.2 stability)). *All Features* dominates overall performance in three out of five models, while *Ext+Multi-CS* and *Multi-CS* excel in solver efficiency and large-scale coverage, respectively. See Appendix for full results.

expansive solution spaces.

For the remaining models, FQA and B2C, we observe consistent patterns (see *Tables 2–3* and *Figures 1b, 1d, 2b, 2c* in Appendix). *Ext+Multi-CS* and *All Features* continue to perform strongly across all metrics, while *Full Pop* reliably enhances stability. In denser models such as B2C, *Extinction* significantly boosts efficiency by accelerating early-stage discovery, though its benefit diminishes in models with less pronounced convergence bottlenecks.

Overall, our results demonstrate that each enhancement contributes complementary benefits. *Multi-CS* consistently improves coverage across all benchmarks. *Extinction* effectively reduces solver effort, particularly in dense or stagnation-prone models. *Full Pop* and *Weighted* configurations enhance stability and robustness, especially when initial population diversity is limited. Combined configurations, such as *All Features* and *Ext+Multi-CS*, offer the best trade-offs across metrics and knowledge base characteristics, supporting their applicability as default strategies for robust and scalable *lazy conflict detection*.

Conclusion

We presented a robust enhancement to *lazy conflict detection* by integrating multi-conflict extraction with genetic diversity control. Our method systematically addresses key limitations of the baseline approach, namely coverage gaps, premature convergence, and result instability, through four targeted mechanisms: *multi-conflict extraction*, *full population reproduction*, *weighted genetic operators*, and *adaptive extinction*. Comprehensive experiments on five real-world configuration knowledge bases demonstrate that our enhanced strategy significantly improves conflict coverage, reduces solver effort, and achieves high result stability. In particular, the *All Features* configuration consistently ranks among the best-performing variants across all evaluation metrics. Further trade-off and convergence analyses further illustrate the scalability and adaptability of our enhancements. Future work will explore adaptive orchestration of enhancement features, i.e., automatically activating mechanisms based on search dynamics, as well as automated parameter tuning to better tailor the approach to different application domains.

Acknowledgments

The work presented in this paper has been developed within the research project GENRE (Generative AI for Requirements Engineering) funded by the Austrian Research Promotion Agency under the project number 915086.

References

Bendík, J.; and Černá, I. 2020. MUST: Minimal Unsatisfiable Subsets Enumeration Tool. In Biere, A.; and Parker, D., eds., *Tools and Algorithms for the Construction and Analysis of Systems*, 135–152. Cham: Springer International Publishing. ISBN 978-3-030-45190-5.

Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; Walsh, T.; et al. 2013. Constraint Acquisition via Partial Queries. In *IJCAI*, volume 13, 475–481.

Carneiro, D.; Novais, P.; and Neves, J. 2013. Using genetic algorithms to create solutions for conflict resolution. *Neurocomputing*, 109: 16–26. New trends on Soft Computing Models in Industrial and Environmental Applications.

Felfernig, A.; Reiterer, S.; Reinfrank, F.-C.; Ninaus, G.; and Jeran, M. 2014. *Conflict Detection and Diagnosis in Configuration*, 73–87. Netherlands: Elsevier B.V., 1 edition. ISBN 978-0-12-415817-7.

Felfernig, A.; Schubert, M.; and Zehentner, C. 2012. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artif. Intell. Eng. Des. Anal. Manuf.*, 26(1): 53–62.

Fleischanderl, G.; Friedrich, G.; Haselboeck, A.; Schreiner, H.; and Stumptner, M. 1998. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4): 59–68.

Greiner, R.; Smith, B. A.; and Wilkerson, R. W. 1989. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1): 79–88.

Heradio, R.; Fernandez-Amoros, D.; Galindo, J. A.; Benavides, D.; and Batory, D. 2022. Uniform and scalable sampling of highly configurable systems. *Empirical Software Engineering*, 27(2): 44.

Jannach, D.; Schmitz, T.; and Shchekotykhin, K. 2015. Parallelized Hitting Set Computation for Model-Based Diagnosis. In *AAAI*, 1503–1510. AAAI Press.

Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for over-Constrained Problems. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, 167–172. AAAI Press.

Junker, U. 2006. Configuration. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*, 837–873. Elsevier.

Katoch, S.; Chauhan, S. S.; and Kumar, V. 2021. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5): 8091–8126.

Le, V. M.; Felfernig, A.; Tran, T. N. T.; and Uta, M. 2024. INFORMEDQX: Informed Conflict Detection for Over-Constrained Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(9): 10616–10623.

Le, V. M.; Vidal Silva, C.; Felfernig, A.; Benavides, D.; Galindo, J.; and Tran, T. N. T. 2023. FASTDIAGP: An Algorithm for Parallelized Direct Diagnosis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(5): 6442–6449.

Mendonca, M.; Branco, M.; and Cowan, D. 2009. S.P.L.O.T.: Software Product Lines Online Tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, 761–762. New York, NY, USA: ACM.

Mitchell, M. 1998. *An introduction to genetic algorithms*. MIT press.

Mittal, S.; and Frayman, F. 1989. Towards a Generic Model of Configuraton Tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, 1395–1401. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Nica, I.; Pill, I.; Quaritsch, T.; and Wotawa, F. 2013. The Route to Success – A Performance Comparison of Diagnosis Algorithms. In *IJCAI'13*, 1039–1045.

Nielsen, J. 1994. *Usability engineering*. Morgan Kaufmann.

O'Sullivan, B.; Nanopulos, A.; Faltings, B.; and Pu, P. 2007. Representative Explanations for Over-Constrained Problems. In *22nd AAAI Conference on Artificial Intelligence*, 323–328. Vancouver, Canada.

Prud'homme, C.; and Fages, J.-G. 2022. Choco-solver: A Java library for constraint programming. *Journal of Open Source Software*, 7(78): 4708.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1): 57–95.

Rodler, P. 2022. A Formal Proof and Simple Explanation of the QuickXplain Algorithm. *Artificial Intelligence Review*, 55(8): 6185–6206.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. ISSN. Elsevier Science. ISBN 9780080463803.

Uran, C.; and Felfernig, A. 2018. Lazy Conflict Detection with Genetic Algorithms. In Mouhoub, M.; Sadaoui, S.; Ait Mohamed, O.; and Ali, M., eds., *Recent Trends and Future Technology in Applied Intelligence*, 175–186. Cham: Springer International Publishing.

Vidal, C.; Felfernig, A.; Galindo, J.; Atas, M.; and Benavides, D. 2021. Explanations for over-constrained problems using QuickXPlain with speculative executions. *Journal of Intelligent Information Systems*, 57(3): 491–508.