A Local Algorithm for Fuzzy Bisimulation (Post-print)

Authors: Hu Jinwei, QIAN Junyan, Qian Junyan

Date: 2023-02-15T00:00:00+00:00

Abstract

To efficiently verify whether given states in fuzzy transition systems satisfy the bisimulation relation, a local algorithm for fuzzy bisimulation is proposed. This algorithm integrates verification with traversal, dynamically expanding the state space while verifying whether states satisfy the fuzzy bisimulation relation, thereby enabling completion of verification by traversing only a partial state space. In certain scenarios, particularly when two states do not satisfy the fuzzy bisimulation relation, the local algorithm for fuzzy bisimulation can more rapidly verify whether given states satisfy the fuzzy bisimulation relation. Both the local algorithm for fuzzy bisimulation and existing global algorithms were implemented in Java, and comparative experiments were conducted. The experimental results demonstrate that, for cases where given state pairs do not satisfy the bisimulation relation, the proposed algorithm achieves higher efficiency than existing global algorithms for fuzzy bisimulation.

Full Text

A Local Algorithm for Fuzzy Bisimulation

Guilin University of Electronic Technology Journal, ChinaXiv Partner Journal, Vol. 43, No. 2, Apr. 2023
School of Computer and Information Security, Guilin University of Electronic Technology, Guangxi

To rapidly verify whether given states in fuzzy transition systems satisfy bisimulation relations, this paper proposes a local algorithm for fuzzy bisimulation that integrates verification with traversal. While verifying whether states satisfy fuzzy bisimulation relations, the algorithm dynamically expands the state space, enabling verification by traversing only a partial state space. When a state does not satisfy the fuzzy bisimulation relation, the local algorithm can

more quickly verify whether the given state satisfies the relation, completing verification using only a portion of the state space.

Abstract

Experimental results demonstrate that when given state pairs do not satisfy bisimulation relations, the proposed algorithm achieves higher efficiency than existing global algorithms for fuzzy bisimulation. **Keywords:** fuzzy transition systems; fuzzy bisimulation; local algorithm. **Document Code:** TP301. **DOI:** 1673-808X(2023)02-0102-06.

We implement both the local algorithm for fuzzy bisimulation and existing global algorithms, conducting comparative experiments. The local algorithm combines verification and traversal, dynamically increasing the state space during verification of whether given states satisfy fuzzy bisimulation. This approach requires traversing only a partial state space, enabling faster verification especially when states do not satisfy fuzzy bisimulation relations.

1 Introduction

Bisimulation, as a fundamental concept for describing behavioral equivalence between systems, serves as an effective means to address the state space explosion problem [**?**]. Paige and Tarjan [**?**] proposed a partition refinement algorithm for bisimulation equivalence class partitioning with time complexity $O(m \log n)$ under the partition refinement paradigm, where n is the number of states and m is the number of transitions. However, this algorithm requires pre-storage of the entire transition system's state space. In many cases, we only need to verify whether specific given states satisfy bisimulation relations. Fernandez et al. [**?**] proposed a local algorithm for bisimulation equivalence verification, which constructs a synchronous product of transition systems and performs depth-first or breadth-first search on successor state pairs. This algorithm has time complexity $O(n^2)$ and simultaneously checks whether each state pair satisfies bisimulation, forming the basis for global algorithms.

Du et al. [**?**] proposed a quasi-local algorithm for bisimulation equivalence verification that lies between local and global algorithms. By adding annotations to states, this approach avoids repeated state visits and reduces time complexity. Lin [**?**] applied local algorithms to value-passing systems, while Deng et al. [**?**] applied local algorithms to probabilistic transition systems. These local algorithms determine whether given states in a transition system satisfy bisimulation relations without computing the entire system's bisimulation equivalence classes.

Fuzzy sets, as extensions of classical sets, primarily describe uncertainty in phenomena and can effectively analyze subjective concepts and uncertain notions [**?**]. Applying fuzzy theory's quantitative analysis to systems enables evaluating the degree to which systems satisfy property specifications. Research on fuzzy

automata [?], Petri nets [?], and fuzzy transition systems [?] has yielded numerous academic results. Notably, when constructing system models using fuzzy automata, the resulting models are not unique. To better determine equivalence between these models, computer scientists introduced bisimulation concepts to fuzzy transition systems.

Petkovic et al. [?] proposed congruence concepts for finite fuzzy automata through algebraic methods. Buchholz [?] and Cao et al. [?] studied bisimulations for finite fuzzy automata, while Cao et al. [?] addressed infinite fuzzy transition systems. Wu et al. [?] characterized fuzzy bisimulations from both algorithmic and logical perspectives. To better characterize behavioral similarity in fuzzy transition systems, Cao et al. [?] introduced behavioral distance concepts. When a fuzzy transition system fully satisfies fuzzy bisimulation properties, Wu et al. [?] provided characterizations of fuzzy bisimulations.

Ignjatovic et al. [?] proposed a global algorithm for fuzzy bisimulation equivalence verification with time complexity $O(c \cdot n^3)$, where c is the number of distinct fuzzy values appearing during computation and n is the number of states in the fuzzy transition system. For fuzzy social networks, [?] presented a fuzzy bisimulation equivalence verification algorithm with time complexity $O(l \cdot n^2)$, where l is the number of distinct fuzzy values and n is the number of nodes in the social network.

Despite these advances in fuzzy bisimulation, research on local algorithms for fuzzy bisimulation remains unexplored. To verify more quickly whether specified states satisfy fuzzy bisimulation relations, this paper adapts algorithmic ideas from [?] and proposes a local algorithm for fuzzy bisimulation equivalence verification, comparing its efficiency with existing global algorithms through experiments.

2 Preliminaries

2.1 Fuzzy Sets and Fuzzy Transition Systems Let S be a non-empty set. A fuzzy set μ on S is a mapping $\mu : S \to [0, 1]$, where $\mu(s)$ represents the membership degree of element s in the fuzzy set. The support set of fuzzy set μ is defined as $\text{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$.

Definition 1 [?]. A fuzzy transition system is a triple $FTS = (S, A, R)$, where: - S is a finite set of states - A is a finite set of actions - $R \subseteq S \times A \times \mathcal{F}(S)$ is the transition relation, where $\mathcal{F}(S)$ denotes the set of all fuzzy sets on S

For any state $s \in S$ and action $\alpha \in A$, we write $s \xrightarrow{\alpha} \mu$ to denote $(s, \alpha, \mu) \in R$, where μ is a fuzzy set on S representing the transition's target distribution.

2.2 Lifted Relations and Bisimulation Given a relation $R \subseteq S \times S$, its lifted relation $R^\dagger \subseteq \mathcal{F}(S) \times \mathcal{F}(S)$ is defined such that $\mu R^\dagger \nu$ holds if and only if: - For all $s \in \text{supp}(\mu)$, there exists $t \in \text{supp}(\nu)$ such that sRt and $\mu(s) \leq \nu(t)$ - For all $t \in \text{supp}(\nu)$, there exists $s \in \text{supp}(\mu)$ such that sRt and $\nu(t) \leq \mu(s)$

Theorem 1 [?]. Let S be a set and $R \subseteq S \times S$ a relation. For any $\mu, \nu \in \mathcal{F}(S)$, if $\mu R^{\dagger} \nu$ holds, then: - $R_{s\rightarrow} = \{t \in S \mid sRt\}$ - $R_{s\leftarrow} = \{t \in S \mid tRs\}$

Definition 2 [?]. A relation $R \subseteq S \times S$ is a fuzzy bisimulation if for any $(s,t) \in R$ and any action $\alpha \in A$: - For all $s \xrightarrow{\alpha} \mu$, there exists $t \xrightarrow{\alpha} \nu$ such that $\mu R^{\dagger} \nu$ - For all $t \xrightarrow{\alpha} \nu$, there exists $s \xrightarrow{\alpha} \mu$ such that $\mu R^{\dagger} \nu$

If a fuzzy bisimulation relation R exists with $(s,t) \in R$, we say states s and t are bisimilar, denoted $s \sim t$.

2.3 Recursive Definition of Fuzzy Bisimulation In fuzzy transition systems, bisimulation can be defined recursively:

Definition 3 [?]. Let $M = (S, A, R)$ be a fuzzy transition system. Define relations $\sim_n \subseteq S \times S$ for $n \in \mathbb{N}$ as: - $s \sim_0 t$ for all $s, t \in S$ - $s \sim_{n+1} t$ iff for all $\alpha \in A$: - For all $s \xrightarrow{\alpha} \mu$, there exists $t \xrightarrow{\alpha} \nu$ such that $\mu \sim_n^{\dagger} \nu$ - For all $t \xrightarrow{\alpha} \nu$, there exists $s \xrightarrow{\alpha} \mu$ such that $\mu \sim_n^{\dagger} \nu$

The fuzzy bisimulation relation is $\sim = \bigcap_{n \in \mathbb{N}} \sim_n$.

Definition 4 [?]. Alternatively, define $\sim' = \bigcap_{n \in \mathbb{N}} \sim'_n$ where: - $s \sim'_0 t$ for all $s, t \in S$ - $s \sim'_{n+1} t$ iff for all $\alpha \in A$: - For all $s \xrightarrow{\alpha} \mu$, there exists $t \xrightarrow{\alpha} \nu$ such that $\mu \sigma m'^{\dagger}_n \nu$ - For all $t \xrightarrow{\alpha} \nu$, there exists $s \xrightarrow{\alpha} \mu$ such that $\mu \sigma m'^{\dagger}_n \nu$

Lemma 1. The fuzzy bisimulation relations defined in Definition 3 and Definition 4 are equivalent: $\sim = \sim'$.

Proof. We prove by induction that $\sim_n = \sim'_n$ for all $n \in \mathbb{N}$. The base case $n = 0$ is trivial. Assume $\sim_n = \sim'_n$. For any $s, t \in S$, $s \sim_{n+1} t$ requires that for all $s \xrightarrow{\alpha} \mu$, there exists $t \xrightarrow{\alpha} \nu$ with $\mu \sim_n^{\dagger} \nu$. By the induction hypothesis, this is equivalent to $\mu \sigma m'^{\dagger}_n \nu$. The symmetric condition follows similarly. Thus $s \sim_{n+1} t$ iff $s \sim'_{n+1} t$, proving $\sim_{n+1} = \sim'_{n+1}$. By induction, $\sim = \sim'$. □

Lemma 2. For any finite fuzzy transition system with n states, the bisimulation relation can be computed in $O(n^2)$ time.

Proof. Let $D = \{\mu(s) \mid s \in S, \mu \in \mathcal{F}(S)\}$ be the finite set of fuzzy values. For any $s \xrightarrow{\alpha} \mu$ and $t \xrightarrow{\alpha} \nu$, we can verify $\mu \sim_n^{\dagger} \nu$ by checking membership conditions across the finite support sets. Since D is finite and the state space is finite, each check requires $O(1)$ time, yielding overall $O(n^2)$ complexity. □

3 Local Algorithm for Fuzzy Bisimulation Verification

3.1 Algorithm Overview Building upon the recursive definition and the algorithm for checking lifted relations $R^{\dagger}$, we present a local algorithm for fuzzy bisimulation equivalence verification. The algorithm combines verification with traversal, dynamically expanding the state space to check only relevant portions.

The algorithm uses four key data structures: - **Errors**: Stores state pairs confirmed to not satisfy fuzzy bisimulation - **Visited**: Stores state pairs that have been traversed - **LetBis**: Stores state pairs assumed to satisfy fuzzy bisimulation (but not yet confirmed) - **Stack**: Stores state pairs pending fuzzy bisimulation verification

When a state pair is visited multiple times but its bisimulation status remains uncertain, we temporarily assume it satisfies the relation and store it in **LetBis**. If subsequent traversal reveals the pair does not satisfy bisimulation, it is moved to **Errors** and the function is re-executed.

3.2 Algorithm Description Algorithm 1: Local Fuzzy Bisimulation Checking

Input: Fuzzy transition systems FTS_1 and FTS_2, and state pair (s, t) to verify
Output: Whether (s, t) satisfies fuzzy bisimulation

```
1. Errors =
2. isBis = unknown
3. while isBis = unknown do
4.   isBis = checkBis(s, t)
5. return isBis
```

Function checkBis(s, t):

```
6.   Visited =
7.   LetBis =
8.   Stack =
9.   Push (s, t) onto Stack
10.  while Stack $\neq$  do
11.    (s', t') = top(Stack)
12.    Visited = Visited  {(s', t')}
13.    if act(s') $\neq$ act(t') then
14.      for each $\alpha$  act(s') do
15.        for each s' $\rightarrow$ _i do
16.          for each t' $\rightarrow$ _j do
17.            if $\alpha$ = act( _i) = act( _j) then
18.              if ( _i,  _j)  Visited then
19.                if check( _i,  _j) = false then
20.                  Errors = Errors  {(s', t')}
21.                  LetBis = LetBis \ {(s', t')}
22.                  return unknown
23.                else
24.                  Push ( _i,  _j) onto Stack
25.                  LetBis = LetBis  {( _i,  _j)}
26.    actionBis =  _{outer} ( _{j} b_{ij})  ( _{i} b_{ij})
27.    isBis =  _{all actions} actionBis
28.    if isBis = false then
```

```
29.      Errors = Errors   {(s, t)}
30.      LetBis = LetBis \ {(s, t)}
31.      return false
32.  if Stack =  then
33.    if isBis = true then
34.      return true
35.  else
36.    return false
```

Data Structures: Fuzzy distributions are stored using linked lists of type `HashTable<state, double>`. The `StatePair` class contains attributes for states s and t, with constructors for initialization. Arrays `Visited`, `LetBis`, and `Errors` are of type `LinkedList<StatePair>`.

Algorithm Flow: When `checkBis(s, t)` executes, `Stack`, `Visited`, and `LetBis` are initialized as empty sets. Each call to `checkBis` resets these local variables. For a successor state pair (s_i, t_j): - If not in `Visited`, it is added to `Visited` - If it fails fuzzy bisimulation, it is added to `Errors` and removed from `LetBis` - If actions differ, traversal continues - If actions match and the pair is already in `Visited`, it is added to `LetBis` - During subsequent traversal, if a pair in `LetBis` is found to violate bisimulation, it is removed from `LetBis` and added to `Errors`, triggering re-execution of `checkBis`

3.3 Termination Proof Theorem 2 (Termination). The local algorithm for fuzzy bisimulation verification always terminates.

Proof. Let $Errors_i$, $LetBis_i$, and $Visited_i$ denote the sets after the i-th execution of `checkBis`. The algorithm terminates when either: 1. `checkBis` returns `true` (bisimulation holds), or 2. `checkBis` returns `false` (bisimulation fails), or 3. No new state pairs are added to `Stack`

Consider the sequence of `Errors` sets: $Errors_0 \subseteq Errors_1 \subseteq \cdots$. Since the state space is finite, this monotonic sequence must stabilize. When $Errors_i = Errors_{i-1}$, all reachable non-bisimilar states have been identified. The algorithm then either confirms bisimulation (returning `true`) or exhausts the search space, ensuring termination. □

3.4 Correctness Proof Theorem 3 (Correctness). For any fuzzy transition systems FTS_1, FTS_2 and states s, t, the algorithm returns `true` iff $s \sim t$.

Proof. Let R_i be the relation induced by $Visited_i$ $Errors_i$ after the i-th `checkBis` execution. We show R_i converges to the fuzzy bisimulation relation $\sim$.

- **Soundness:** If the algorithm returns `true`, then for all visited pairs (s', t'), all successor pairs satisfy the bisimulation conditions. By construction, this means $\mu R_i^{\dagger} \nu$ for all relevant transitions, implying $s \sim t$.

- **Completeness:** If $s \sim t$, then by definition there exists a bisimulation relation R containing (s,t). The algorithm explores all reachable pairs from (s,t). Since R is a bisimulation, no pair in R will be added to `Errors`. Thus, the algorithm will never return `false` and will eventually return `true`.

The key invariant is that $R_i \subseteq \sim$ for all i, with R_i growing monotonically. Upon termination, either $R_k = \sim$ for some k (returning `true`) or a violation is found (returning `false`). □

4 Complexity Analysis

4.1 Data Structures Fuzzy transition systems are stored using adjacency lists. The storage structure uses `HashTable` to map state-action pairs to fuzzy distributions, maintaining consistent data types. For state pair storage, we define a `StatePair` class containing states s and t, with value types matching the outermost fuzzy distribution type (set to `double`).

4.2 Time Complexity **Theorem 4.** The local algorithm has time complexity $O(n^2m^2)$, where n is the number of states and m is the number of transitions.

Proof. The worst-case occurs when `checkBis` traverses all state pairs and all transitions. For fuzzy transition systems FTS_1 and FTS_2 with n states and m transitions each: - The outer `while` loop executes at most n times - The inner traversal may process up to n^2 state pairs - For each pair, checking all actions and transitions requires $O(m^2)$ time - The lifted relation check $\mu R^{\dagger} \nu$ processes at most $|\text{supp}(\mu)| \cdot |\text{supp}(\nu)| \leq m^2$ pairs

By the multiplication principle, total time complexity is $O(n^2m^2)$. When only verifying a single transition system ($FTS_1 = FTS_2$), this reduces to $O(n^2m)$. □

4.3 Space Complexity **Theorem 5.** The local algorithm has space complexity $O(n^2)$.

Proof. In the worst case, `Visited` stores all traversed state pairs, with at most n^2 pairs. Since `LetBis` and `Errors` are subsets of visited pairs, the total space is bounded by $O(n^2)$. The adjacency list representation requires $O(n+m)$ space, dominated by $O(n^2)$ for the pair sets. □

5 Experimental Evaluation

5.1 Experimental Setup We compare the local algorithm's practical efficiency against the global algorithm [**?**]. Since benchmark fuzzy transition systems are unavailable, we randomly generate systems with varying states and transitions. Experiments were conducted on a Windows 10 system with an Intel Core i7-7700HQ processor at 2.8 GHz and 8 GB RAM, implemented in Java.

5.2 Results and Analysis **Table 1:** Runtime comparison for systems with same state count but different transition counts

States	Transitions	Local Algorithm (s)	Global Algorithm (s)	Bisimulation?
90	243	0.068	0.151	False
90	252	0.167	0.748	False

Table 2: Runtime comparison for systems with different state and transition counts

States	Transitions	Local Algorithm (s)	Global Algorithm (s)	Bisimulation?
90	243	0.275	0.405	True
90	252	0.748	1.234	True

Observations: 1. **Efficiency:** The local algorithm consistently outperforms the global algorithm, particularly when state pairs do not satisfy bisimulation (False cases), matching theoretical predictions. 2. **Scalability:** Although the global algorithm has lower asymptotic complexity, the local algorithm's depth-first traversal with on-the-fly verification proves more efficient for targeted state pair checking. 3. **Consistency:** Runtime differences between satisfying and violating cases are minimal for the local algorithm, unlike global algorithms that must explore the entire state space regardless.

The local algorithm's advantage stems from its ability to: - Traverse only reachable portions of the state space - Terminate early upon discovering violations - Avoid pre-computing the entire system's bisimulation partition

6 Conclusion and Future Work

This paper presents a local algorithm for fuzzy bisimulation equivalence verification with time complexity $O(n^2m^2)$ and space complexity $O(n^2)$. Experimental results confirm its superior efficiency over global algorithms [**?**] when verifying specific state pairs, particularly for non-bisimilar states.

Future Work: We plan to extend this approach to **weak bisimulation** for fuzzy transition systems. Weak bisimulation matches a given process transition with another process's transition sequence, offering another effective solution to state space explosion. Future research will define fuzzy weak bisimulation and develop corresponding local and quasi-local verification algorithms.

References

[1] Baier C, Katoen J P. *Principles of Model Checking*. Cambridge: MIT Press, 2008: 449-582.

[2] Paige R, Tarjan R. Three partition refinement algorithms. *SIAM Journal on Computing*, 1987, 16(6): 973-989.

[3] Fernandez P. *Verifying Bisimulations "On-the-Fly"*. In: Proceedings of International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols. New York, NY: ACM, 1990: 95-110.

[4] Du W, Deng Y. Distribution-based quasi-local algorithm for checking bisimilarity. In: *IEEE International Conference on Computer Science and Engineering*. Piscataway, NJ: IEEE Press, 2011: 1-5.

[5] Lin M. Local algorithm for bisimilarity in value-passing systems. In: *Fourth International Conference on Frontier of Computer Science and Technology*. Piscataway, NJ: IEEE Press, 2009: 401-407.

[6] Deng C, Palamidessi C. Local algorithm for probabilistic bisimilarity checking. In: *International Conference on Concurrency Theory*. Springer, 2009: 1-15.

[7] Zadeh L A. Fuzzy sets. *Information and Control*, 1965, 8(3): 338-353.

[8] Doostfatemeh M, Kremer S C. New directions in fuzzy automata. *International Journal of Approximate Reasoning*, 2005, 38(2): 175-214.

[9] Cao Y, Ezawa Y. Nondeterministic fuzzy automata. *Information Sciences*, 2012, 191: 86-97.

[10] Li Y. Fuzzy Turing machines: variants and universality. *IEEE Transactions on Fuzzy Systems*, 2008, 16(6): 1491-1502.

[11] Bugarin J, Barro S. Fuzzy reasoning supported by Petri nets. *IEEE Transactions on Fuzzy Systems*, 1994, 2(2): 135-150.

[12] Pedrycz W, Gomide F. A generalized fuzzy Petri net model. *IEEE Transactions on Fuzzy Systems*, 1994, 2(4): 295-301.

[13] Wu H, Chen Y. Coalgebras of fuzzy transition systems. *Electronic Notes in Theoretical Computer Science*, 2014, 301: 91-101.

[14] Pan H, Zhang M, Chen Y. Bisimilarity of doubly labeled fuzzy transition systems. *Quantitative Computing*, 2012: 207-214.

[15] Wu T, Han H, Chen Y. Bisimulations and homomorphisms of fuzzy transition systems revisited. *International Journal of Approximate Reasoning*, 2018, 99: 1-11.

[16] Cao Y, Chen G, Kerre E. Bisimulations for fuzzy-transition systems. *IEEE Transactions on Fuzzy Systems*, 2011, 19(3): 540-552.

[17] Petkovic T. Congruences and homomorphisms of fuzzy automata. *Fuzzy Sets and Systems*, 2006, 157(3): 444-458.

[18] Buchholz M. Bisimulation relations for weighted fuzzy automata. *Theoretical Computer Science*, 2008, 393(1-3): 109-123.

[19] Sun Y, Yang W. Bisimulation relations for fuzzy finite automata. *Fuzzy Mathematics*, 2009, 23: 92-100.

[20] Cao X, Wang Y, Sun Y. Behavioral distances for fuzzy-transition systems. *IEEE Transactions on Fuzzy Systems*, 2012, 21(4): 735-747.

[21] Wu Y, Chen Y, et al. Algorithmic and logical characterizations of fuzzy bisimulations. *Fuzzy Sets and Systems*, 2018, 333: 106-123.

[22] Ignjatovic J, Ciric M, Damljanovic N. Computation of the greatest simulations and bisimulations between fuzzy automata. *Fuzzy Sets and Systems*, 2012, 208: 22-42.

[23] Ignjatovic J, Ciric M, Stankovic I. Bisimulation analysis of fuzzy social networks. In: *Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology*. Gijón: Atlantis Press, 2015: 404-411.

[24] Qian J, Zha X. Symbolic reachability analysis of concurrent programs based on automata. *Journal of Guilin University of Electronic Technology*, 2013, 33(4): 300-304.

[25] Qian J. Construction method for Büchi monitor automata simplification. *Journal of Guilin University of Electronic Technology*, 2019, 39(5): 374-378.

Note: Figure translations are in progress. See original paper for figures.