وارسی ویژگی دسترس‌پذیری در سامانه‌های نرم‌افزاری پیچیده و هم‌روند با استفاده از الگوریتم‌های جستجوی هوشمند

جعفر پرتابیان[۱]، وحید رافع[۲*]، حمید پروین[۳]، صمد نجاتیان[۴] و کرم‌الله باقری‌فرد[۵]

[۱]دانشکده مهندسی کامپیوتر، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران

[۲]دانشکده فنی و مهندسی، گروه مهندسی کامپیوتر، دانشگاه اراک، اراک، ایران

[۳]دانشکده مهندسی کامپیوتر، واحد نورآباد ممسنی، دانشگاه آزاد اسلامی، نورآباد ممسنی، ایران

[۴و۵]دانشکده فنی و مهندسی، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران

[۴]عضو باشگاه پژوهشگران جوان و نخبگان، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج ایران

[۳]عضو باشگاه پژوهش‌گران جوان و نخبگان، واحد نورآباد ممسنی، دانشگاه آزاد اسلامی، نورآباد ممسنی، ایران

چکیده

روش وارسی مدل، روشی رسمی و مؤثر جهت تأیید سامانه‌های نرم‌افزاری است که با تولید و بررسی همه حالت‌هایِ ممکنِ مدلی از سامانه نرم‌افزار به تحلیل آن می‌پردازد. در سامانه‌های ایمنی– بحرانی، نمی‌توان ریسک بروز خطا را حتی در فرآیند تست پذیرفت و لذا لازم است فرآیند درستی‌یابی، قبل از پیاده‌سازی و در سطح مدل انجام شود. استفاده از این روش به‌منظور بررسی خواصی مانند ایمنی ایجاب می‌کند که تمام حالت‌های قابل دسترس (تمام فضای حالت) تولید و سپس فضای حالت سامانه مورد نظر به‌صورت دقیق بررسی شوند. چالش اساسی روش وارسی مدل در سامانه‌های بزرگ و پیچیده که دارای فضای حالت گسترده و نامحدود هستند، مشکل انفجار فضای حالت (کمبود حافظه در تولید همه حالت‌های ممکن) است. سامانه‌های تبدیل گراف، از پرکاربردترین سامانه‌های مدل‌سازی رسمی و راه‌کاری مناسب به‌منظور مدل‌سازی و وارسی سامانه‌های پیچیده هستند. در سامانه‌هایی که تأیید ویژگی ایمنی غیرممکن است، می‌توان با جستجویِ یک حالت قابل دسترسی که در آن پیکربندی خاصی (به‌عنوان مثال خطا یا رفتار نامطلوب) رخ می‌دهد، ویژگی ایمنی را رد کرد. مطالعات اخیر حاکی از آن است که اکتشاف جزئی و هوشمندانه بخشی از فضای حالت می‌تواند راه حل مناسبی برای مشکل انفجار فضای حالت باشد. هدف این پژوهش، استفاده از الگوریتم جنگل تصادفی در وارسی مدل است که می‌تواند با گزینش تعداد محدودی مسیر امیدبخش مشکل انفجار فضای حالت را برطرف سازد. مسیری امیدبخش است که احتمال رسیدن به یک جواب از طریق این مسیر، بیشتر از بقیه مسیرها باشد. در روش پیشنهادی، ابتدا مدل کوچکی از سامانه با استفاده از زبان رسمی سامانه توصیف گراف (GTS) ایجاد، سپس، از فضای حالت مدل کوچک، مجموعه آموزشی از مسیرهایی که به هدف می‌رسند ایجاد می‌شود. پس‌ازآن، مجموعه آموزشی تولیدشده در اختیار الگوریتم جنگل تصادفی قرار داده می‌شود تا روابط منطقی موجود در آن شناسایی و کشف شوند. در مرحله بعد از دانش به‌دست‌آمده جهت پیمایش هوشمند و غیر کامل فضایِ حالتِ مدلِ بزرگ استفاده می‌شود. رویکرد پیشنهادی برای تأیید ویژگی دسترس‌پذیری و رد ویژگی ایمنی در سامانه‌های بزرگ و پیچیده که ایجاد تمام فضای حالت سامانه ناممکن است، استفاده می‌شود. به منظور ارزیابی رویکرد پیشنهادی، این رویکرد در ابزار GROOVE که از ابزار متن‌باز برای طراحی و وارسی مدل برای سامانه‌های تبدیل گراف است، اجرا شده است. نتایج نشان می‌دهند که روش پیشنهادی ازنظر میانگین زمان اجرا و طول شاهد تولیدشده نسبت به روش‌های مورد مقایسه عملکرد بهتری دارد.

واژگان کلیدی: وارسی مدل، تأیید سامانه‌های نرم‌افزاری، کشف دانش، انفجار فضای حالت، جستجوی هوشمند

Reachability checking in complex and concurrent software systems using intelligent search methods

Jaafar partabian[1], Vahid Rafe[2*], Hamid Parvin[3], Samad Nejatian[4] & Karamollah Bagherifard[5]

* Corresponding author

*نویسندهٔ عهده‌دار مکاتبات

• تاریخ ارسال مقاله: ۱۳۹۸/۷/۵ • تاریخ پذیرش: ۱۳۹۹/۱۲/۱۱ • تاریخ انتشار: ۱۴۰۱/۰۳/۳۱ • نوع مطالعه: پژوهشی

[DOI: 10.52547/jsdp.19.1.153]

[1]Ph.D. Candidate, Department of Computer Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran
[2]Department of Computer Engineering, Faculty of Engineering, Arak University, Arak, Iran.
[3]Department of Computer Engineering, Nourabad Mamasani Branch, Islamic azad University, Nourabad Mamasani, Iran
[4,5]Department of Electrical Engineering Yasooj Branch, Islamic azad University, Yasooj, Iran
[4]Young Researchers and Elite club Yasooj branch, Islamic Azad University, Yasooj, Iran
[3]Young Researchers and Elite Club Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran

Abstract

The model checking technique is a formal and effective method for verifying software systems, which analyses it via generating and examining all possible states of a model of the software system. In safety-critical systems, one could not admit the risk of error even in the testing process, therefore it is necessary to carry out the verification process before implementation and at the model level. Using this technique to evaluate properties such as security entails all available states (all state space) being generated, then the state space of the system in question be carefully examined. The main challenge of the model checking technique in large and complex systems with wide or infinite state space is the problem of state space explosion (lack of memory in the generation of all possible states). Graph transformation systems are one of the most widely used formal modeling systems and a suitable solution for modeling and checking complex systems. In systems where security property verification is not possible, the security feature can be refuted by searching for an accessible mode in which a specific configuration (e.g. error or undesirable behavior) occurs. Recent studies advocate that partial and intelligent exploration of part of the state space could be a good solution to the problem of state space explosion. The goal of this study is to use the random forest algorithm in the model checking which can solve the problem of state space explosion by selecting a few promising paths. A path is hopeful whenever the probability of reaching an answer through this path is higher than other paths. In the proposed method, a small model of the system is first created using the official language of the Graph Description System (GTS). Afterwards, a training data set of paths to the goal is generated from the small model mode space. The generated training data set is then provided to the random forest algorithm to identify and discover the logical relationships within it. In the next stage, the acquired knowledge is used to intelligently explore the incomplete space of the large model state. The proposed approach is used in the verification of the reachability property and to refute the safety feature in large and complex systems where it is impossible to generate the entire system state space. In order to evaluate the proposed approach, it has been implemented in GROOVE which is an open source tool for designing and checking models in graph conversion systems. The results indicate that the proposed method performs better than the compared methods in terms of average running time and the length of the generated witness.

Keywords: Software systems verification, Knowledge discovery, State space explosion, intelligent search

.

۱- مقدمه

همزمان با افزایش به‌کارگیری سامانه‌های نرم‌افزاری در زندگی روزمرهٔ بشر، نیاز به تولید سامانه‌های بدون خطا روزبه‌روز بیشتر احساس می‌شود. تولید سیستم‌های مطمئن و عاری از خطا به‌ویژه در کاربردهای بحرانی - حیاتی که بروز هرگونه خطایی منجر به ازبین‌رفتن جان انسان‌های زیادی می‌شود، اهمیت بالایی دارد. عملیات درستی یابی به‌خصوص در سامانه‌های بحرانی- حیاتی، باید در مرحله تحلیل و طراحی و قبل از مرحله پیاده‌سازی انجام شود. یکی از رایج‌ترین روش‌های درستی‌یابی رسمی، روش وارسی مدل است. وارسی مدل به‌دنبال آن است که آیا مدل یک سامانه، نیازمندی‌های از قبل تعیین‌شده را برآورده می‌کند یا خیر؟

سامانه تبدیل گراف، یکی از روش‌های پرکاربرد برای مدل‌سازی سامانه است. سامانه تبدیل گراف امکان توصیف رفتار و حالت‌های سامانه به‌صورت رسمی و در قالب گراف‌ها و نمودارها را فراهم می‌کند. اگرچه وارسی مدل، روشی مؤثر و کارآمد جهت کشف خطاهای سامانه است؛ اما نیاز به تولید تمام فضای حالت مدل سامانه، فرایند وارسی مدل را با مشکل انفجار فضای حالت روبرو می‌کند؛ به‌طوری‌که با بزرگ‌شدن ابعاد مسأله، فضای حالت به‌صورت نمایی رشد می‌کند و حافظه سیستمِ وارسی‌کنندهٔ مدل، قادر به تولید، نگهداری و بررسی همه حالات نیست؛ درنتیجه، فرآیند وارسی مدل به‌دلیل کمبود حافظه اغلب بدون نتیجه و ناموفق است. در سال‌های گذشته برای کاهش مشکل انفجار فضای حالت، راه‌حل‌هایی ابداع شده است. ازجمله این روش‌ها عبارت‌اند

[DOI: 10.52547/jsdp.19.1.153]

از: روش وارسی مدل نمادین [1] که به‌کمک نمودارهای تصمیم‌گیری سعی در کاهش حجم و فشرده‌سازی مدل‌دارد. روش کاهش تقارنی [2] با ادغام حالت‌هایی از مدل که شباهت ساختاری باهم دارند، سعی در ایجاد مدل کوچک‌تری دارد. روش کاهش بر مبنای ترتیب جزئی [3] که برای ساده‌سازی مدل‌هایی که از توازی چند مدل به‌وجود آمده‌اند، استفاده می‌کند؛ به‌طوری‌که می‌توان با تغییر ترتیب اجرای رویه‌های مستقل از هم، بعضی از حالات را حذف کرد. روش وارسی سناریومحور [4] در مدل‌های مبتنی بر سامانه تبدیل گراف، به حذف حالت‌ها و گذرهایی که در سناریو تأثیری ندارند، می‌پردازد و پس‌ازآن، اقدام به بررسی درستی یا نادرستی سناریو می‌کند. روش روش انتزاعی [5] ساختارهای مشابه در فضای حالت مدل را باهم ادغام می‌کند و سعی در کاهش اندازه فضای حالت مدل دارد. همه روش‌های بالا سعی در کاهش اندازه فضای حالت مدل دارند؛ ولی در هیچ‌کدام از آن‌ها مکاشفه‌ای در وارسی مدل وجود ندارد.

اگرچه روش‌های یادشده، مشکل انفجار فضای حالت را کاهش می‌دهند، اما همچنان فرایند وارسی مدل در سامانه‌های پیچیده، از مشکل کمبود حافظه و سرعت‌پایین رنج می‌برد.

مابقی مقاله به این صورت سازمان‌دهی می‌شود: بخش ۲ شامل کارهای مرتبط است. رویکرد پیشنهادی در بخش ۳ ارائه می‌شود. در بخش ۴ به مفاهیم اولیه ازجمله روش وارسی مدل، سامانه تبدیل گراف و جنگل تصادفی پرداخته می‌شود. پیاده‌سازی و ارزیابی روش پیشنهادی در بخش ۵ آورده می‌شود. بخش ۶ به نتیجه‌گیری مقاله اختصاص می‌یابد.

۲- کارهای مرتبط

در سال‌های اخیر از روش‌های متفاوتی برای وارسی مدل استفاده شده است. در [6] برای وارسی ویژگی ایمنی سامانه از الگوریتم *A* و* *IDA* استفاده شده است. در این پژوهش، فاصله همینگ بین حالت فعلی و حالت هدف به‌عنوان تابع مکاشفه‌ای در نظر گرفته شده و سپس در ابزار *HSF-SPIN* [7] پیاده‌سازی و مورد ارزیابی قرارگرفته است. نتایج آزمایش‌ها نشان می‌دهند که این روش قادر است با کاوش تعداد کمتری از حالت‌ها نسبت به ابزار *SPIN* به وارسی ویژگی ایمنی در سامانه دست یابد.

در [8] برای پرهیز از کاوش حالت‌های غیرضروری، ترکیبی از الگوریتم‌های جستجوی نخست سطح و الگوریتم *A** روشی به نام *BDDA** ارائه شده است. نتایج، نشان از کارایی بالاتر این روش در مقایسه با الگوریتم‌های جستجوی نخست سطح و نخست عمق دارد.

در [9] با افزودن امکان عقب‌گرد به الگوریتم جستجوی نخست عمق، روش *DFHS* جهت وارسی ویژگی ایمنی سامانه ایجاد شده است. روش *DFHS* در وارسی‌کنندهٔ مدل *Java Path Finder (JPF)* [10] پیاده‌سازی شده و مورد ارزیابی قرارگرفته است. نتایج به‌دست‌آمده نشان می‌دهد این روش در مقایسه با روش‌های جستجوی نخست عمق و نخست سطح عملکرد بهتری دارد.

در [11,12] مکاشفه‌ای برای به‌کارگیری در روش‌های *A** و جستجوی اول عمق برای وارسی ویژگی دسترس‌پذیری در سامانه‌های مدل‌شده با سامانه تبدیل گراف ارائه شده است. تابع مکاشفه‌ای یادشده بر مبنای شباهت ساختاری بین گراف متناظر با حالت فعلی و گراف متناظر با حالت نهایی طراحی‌شده است. نتایج پژوهش، نشان می‌دهد روش یادشده، از عملکرد بهتری در مقایسه با روش‌های جستجوی نخست عمق و جستجوی نخست سطح برخوردار است.

در [13,14] نویسندگان به‌منظور کاهش اندازه فضای حالت به تابعی مکاشفه‌ای رسیده‌اند که با به‌کارگیری در روش‌های جستجوی *A**، جستجوی نخست عمق و تپه‌نوردی به وارسی ویژگی دسترس‌پذیری پرداخته است. آزمایش‌ها نشان از عملکرد بهتر در مقایسه با روش‌های قبلی [11,12] دارد.

در [15] راه‌حلی برای کشف خطای بن‌بست در سامانه‌های واکنشی ارائه شده است. در این راه‌حل با کمک الگوریتم ژنتیک به‌جای جستجوی کامل فضای حالت، جستجو به سمت حالت‌های خطا هدایت می‌شود. این روش در ابزار *VERISOFT* [16]، از ابزارهای جستجوی فضای حالت سامانه، پیاده‌سازی و آزمایش‌شده است. نتایج نشان می‌دهد، این روش نسبت به روش‌های تصادفی در زمان کوتاه‌تری می‌تواند حالت‌های خطا در سامانه کشف کند.

در [17] روشی مبتنی بر ژنتیک در کشف خطای بن‌بست در مدل‌های مبتنی بر سامانه تبدیل گراف ارائه شده است. در این روش هر کروموزوم شامل مسیری با اندازه معین در فضای حالت است. با به‌کارگیری عملگرهای جهش و آمیزش در تولید کروموزوم‌های نسل بعدی، سعی در کشف مسیری متناظر با حالت بن‌بست

[DOI: 10.52547/jsdp.19.1.153]

دارد. چنانچه چنین مسیری یافت شود، به‌عنوان‌مثال نقض اعلام می‌شود. نتایج آزمایش‌های انجام‌شده نشان می‌دهد این روش در بعضی از مدل‌های بزرگ موفق بوده است؛ اما همچنان در مدل‌های خیلی بزرگ و پیچیده با شکست مواجه شده است.

در [18] روشی برای تشخیص خطای بن‌بست در سامانه‌های توصیف‌شده با زبان رسمی تبدیل گراف ارائه شده است. این روش برای اجتناب از گرفتارشدن در دام بهینه محلی، از ترکیب الگوریتم‌های پرندگان و جستجوی گرانشی به‌منظور جستجو فضای حالت استفاده می‌شود. نتایج ارزیابی این روش در ابزار *GROOVE* [19]، نشان می‌دهد این روش از جستجوی نخست عمق و جستجوی نخست سطح، نیز نسبت به روش‌های مبتنی بر الگوریتم ژنتیک سریع‌تر بوده و دقت بالاتری دارد.

در [21] راه‌حلی به نام *BAPSO* به‌کمک الگوریتم‌های خفاش و پرندگان برای تشخیص حالت بن‌بست در سامانه‌های نرم‌افزاری ارائه شده است. ارزیابی این روش در ابزار *GROOVE*، بیان‌گر آن است که روش یادشده، عملکرد مناسب‌تری نسبت به هرکدام از الگوریتم‌های پرندگان و خفاش دارد؛ اما در سامانه‌های پیچیده با شکست مواجه می‌شود.

در [23-21]، روش‌های مبتنی بر کلونی مورچگان برای کشف حالت خطا در فرایند وارسی مدل طراحی‌شده است. با توجه به اینکه الگوریتم مورچگان بر مبنای تلاش مورچه‌ها در یافتن غذا در کوتاه‌ترین مسیر استوار است؛ بنابراین در این روش نیز با تولید مسیرهای کوتاه‌تر کشف خطا نسبت به مسیرهای طولانی‌تر، فضای کمتری برای ذخیره حالت‌ها به کار گرفته می‌شود. این روش‌ها توانسته‌اند بهینه‌ترین پاسخ یا پاسخ نزدیک به بهینه را بیابند.

در [24] با به‌کارگیری یادگیری تقویتی مبتنی بر جایزه و جریمه، راه‌حل جدیدی ارائه شده است. این روش در وارسی ویژگی پیشروی در مدل وارسی *on-the-fly* [25] به کار گرفته شده است. در مدل وارسی *on-the-fly*، برخلاف وارسی معمولی، عملیات وارسی مدل، هم‌زمان با پیمایش فضای حالت صورت می‌گیرد. نتایج نشان از عدم دقت کافی و سرعت پایین این روش در مقایسه با روش‌های فرامکاشفه‌ای دارد.

در [26] نویسندگان با ارائه مکاشفه مبتنی بر روش داده‌کاوی به وارسی ویژگی‌های ایمنی، زنده‌بودن و قابلیت دسترسی در سامانه‌های نرم‌افزاری پیچیده می‌پردازند. در این روش نویسندگان با کشف الگوهای تکراری از مدل کوچک مسأله، توانسته‌اند به مثال نقض ویژگی‌های گفته‌شده در مدل بزرگ مسأله دست پیدا کنند. اگرچه این روش از سرعت و دقت بالاتری نسبت به روش‌های مکاشفه‌ای برخوردار است، اما به مدل کوچک مسأله وابسته بوده و نیاز به تنظیم پارامترهای اولیه در تابع کشف الگوهای مکرر است.

در [27] با کمک روش یادگیری ماشین، روشی جهت انکار ویژگی‌های ایمنی، پیشروی و تأیید ویژگی دسترس‌پذیری ایجاد شده است. در این روش، هم‌زمان با پیمایش فضای حالت، به‌کمک شبکه بیزین وابستگی‌های بین قوانین موجود در فضای حالت، استخراج می‌شود و در ادامه سعی در بهبود شبکه بیزین ایجادشده می‌کند. سپس با دانش کشف‌شده از شبکه بیزین حاصل، بقیه فضای حالت را پیمایش می‌کند. نتایج آزمایش‌ها در ابزار *GROOVE* نشان‌دهنده کارایی بهتر این روش در مقایسه با روش‌های تکاملی و فراابتکاری است. این روش وابسته به انتخاب بخشی از فضای حالت است که جهت یادگیری وابستگی‌های موجود در فضای حالت استفاده می‌شود.

در [28] با کمک روش یادگیری ماشین و کشف مسیرهای امیدبخش به‌وسیلهٔ الگوریتم یادگیر ماشین *AdaBoost* در مدل کوچک مسأله و سپس پیمایش این مسیرها در مدل بزرگ مسأله، توانسته است بر مشکل انفجار فضای حالت غلبه کند. در مقایسه با روش‌های تکاملی و هیوریستیک نیاز به پیمایش حالت‌های کم‌تری در فضای حالت جهت رسیدن به حالت هدف دارد.

در [29] نویسندگان با استفاده از روش *n-gram* ضمن مدیریت فضای حالت مسأله در سامانه‌های پیچیده و بزرگ، به دقت بالاتری در کشف خطای بن‌بست دست پیدا کرده‌اند. در این روش فضای به‌نسبه زیادی برای ذخیره جدول *n-gram* لازم است؛ بنابراین ازنظر مصرف حافظه بهینه نیست.

اگرچه روش‌های بالا توانسته‌اند مشکل انفجار فضای حالت را تا حدودی مدیریت کنند، اما دقت و سرعت هم‌گرایی راه‌حل‌های ارائه‌شده، به‌ویژه در سامانه‌های بزرگ و پیچیده هنوز پایین است. همچنین، اغلب راه‌حل‌ها فقط به تشخیص بن‌بست پرداخته‌اند؛ درصورتی‌که انواع پیچیده‌تری از ویژگی‌ها ازجمله پیشروی و دسترس‌پذیری هنوز باقی‌مانده‌اند.

در این مقاله سعی شده است با هوشمند سازی روش وارسی مدل به‌کمک الگوریتم یادگیری ماشین

[DOI: 10.52547/jsdp.19.1.153]

نظارت‌شده، علاوه بر دست‌یابی به کارایی بالاتر نسبت به روش‌های قبلی، امکان وارسی ویژگی دسترس‌پذیری نیز فراهم شود. در روش پیشنهادی ابتدا دانش لازم در مورد به‌کارگیری قوانین (اعمالی که باعث تغییر حالت سامانه می‌شوند) با استفاده از الگوریتمِ یادگیریِ ماشین کسب‌شده، سپس با استفاده از دانش کسب‌شده، فضای حالت مدل به‌طور هوشمندانه پیمایش شود.

۳- رویکرد پیشنهادی

راه‌حل پیشنهادی در قالب سه فاز طراحی شده است.

۳-۱- فاز نخست: وارسی فضای حالت مدلِ کوچک

در این فاز مدل کوچکی از سامانه ایجاد و فضای حالت آن به‌منظور بررسی ویژگی دسترس‌پذیری به‌طور کامل پیمایش می‌شود. برای دست‌یابی به هدف یادشده، عملیات زیر در این فاز انجام می‌شود:

۳-۱-۱- مدلی از سامانه در قالب گراف نوع به‌صورت خودکار و یا به‌صورت دستی توسط طراح ایجاد می‌شود.

۳-۱-۲- تمام فضای حالت مدل کوچک، در قالب گراف ایجاد می‌شود. در این گراف، گره‌ها بیانگر حالت‌ها و یال‌ها بیان‌گر قوانین تبدیل (قوانین تبدیل، حالتِ سیستم را از یک حالت به حالت دیگری تغییر می‌دهند) هستند. فضای حالت به‌طور کامل با استفاده از الگوریتم *BFS* جستجو می‌شود تا مسیرهای منتهی به حالت هدف از بقیه مسیرها شناسایی شوند. (حالتی که در آن ویژگی دسترس‌پذیری برقرار باشد، به‌عنوان حالت هدف در نظر گرفته می‌شود). جستجوی *BFS* در الگوریتم (۱) آورده شده است.

Algorithm 1 : The BFS algorithm in GROOVE

```
1-  Input: M: a model described by GTS.
2-  Output: S: the state space of M.
3-  GraphState state = the initial state of M;
4-  LinkedList<GraphState> stateQueue=new LinkedList<GraphState> ();
5-  stateQueue.enqueue (state);
6-  S.nodes.add (state);
7-  while stateQueue.size () > 0 do
8-      state = stateQueue.dequeue ();
9-      for each  MatchResult match in state.getMatches () do
10-             GraphState next = state.applyMatch(match);
11-             if next != null then
12-                 if !S.nodes.contains (next) then
13-                     stateQueue.enqueue (next);
14-                     S.nodes.add (state);
15-                 end if
16-                 S.edges.add (new Transition (state,match,next));
17-             end if
18-     end for
19- end while
20- return S;
```

۳-۲- فاز دوم: آموزش و یادگیری قوانین حاکم بر فضای حالت مدلِ کوچک

در این فاز یک مجموعهٔ آموزشی از مسیرهای پیمایش شده ایجاد می‌شود. مجموعهٔ آموزشی به الگوریتم یادگیر ماشین آموزش داده می‌شود تا روابط منطقی و دانش نهفته در آن کشف شود. عملیات زیر در این فاز صورت می‌گیرد.

۳-۲-۱- کلیه مسیرهای موجود در گراف استخراج می‌شوند. هر مسیر به‌صورت $s_0 r_0 s_1 r_1 \dots r_{l-1} s_l$ تولید می‌شود که s_i بیان‌گر حالت iام و r_i نشان‌دهنده قانون iام است. s_0 به‌عنوان حالت اولیه و s_l به‌عنوان حالت موجود در آخرین سطح گراف در نظر گرفته شده است؛ سپس با حذف حالت‌ها از هر مسیر، دنباله‌ای از قوانین حاصل می‌شود. هر دنباله به‌عنوان یک نمونهٔ آموزشی

[DOI: 10.52547/jsdp.19.1.153]

برچسب‌دار در مجموعهٔ آموزشی قرار می‌گیرد. (مسیری که ویژگی دسترس‌پذیری در آن برقرار باشد، برچسب ۱ و مسیری که در آن ویژگی دسترس‌پذیری برقرار نباشد برچسب صفر زده می‌شود).

۳-۲-۲- مجموعه آموزشی در اختیار الگوریتم جنگل تصادفی قرار داده می‌شود تا روابط حاکم بر فضای حالت را کشف و دانش متناظر با آن را کسب کند (عملیات آموزش و یادگیری مدل).

۳-۳- فاز سوم: پیمایش هوشمندانه مدلِ بزرگ

در این فاز فضای حالت مدل بزرگ به‌طور غیر کامل و هوشمندانه مورد پیمایش قرار می‌گیرد. دانش به‌دست‌آمده در مرحله قبل را می‌توان جهت وارسی ویژگی دسترس‌پذیری به‌صورت کارآمد در مدل بزرگ مورداستفاده قرارداد. عملیات زیر در این فاز صورت می‌گیرد.

۳-۳-۱- فضای حالت مدل بزرگ تا سطح مشخصی پیمایش می‌شود (سطح پیمایش متناسب با ظرفیت حافظه تعیین می‌شود). مسیرهای استخراج‌شده مطابق آنچه در ۳-۲-۱ آمده است، در مجموعه آزمایشی قرار داده می‌شود. مجموعهٔ آموزشی را در اختیار الگوریتم جنگل تصادفی قرار داده تا با توجه به دانش کسب‌شده از فضای حالت مدل کوچک، برچسب مسیرها را پیش‌بینی کند. مسیرهایی که برچسب یک دریافت کنند، به‌عنوان مسیرهای امیدبخش در فضایِ حالتِ مدلِ بزرگ، در نظر می‌گیریم.

۳-۳-۲- برای اجتناب از گرفتارشدن در دام انفجار فضای حالت، فقط مسیرهای امیدبخش پیمایش می‌شوند و در صورت وجود حالت هدف، مسیر موردنظر به‌عنوان شاهد جهت تأیید ویژگی دسترس‌پذیری نشان داده می‌شود. الگوریتم (۲) چگونگی تولید شاهد را نشان می‌دهد.

Algorithm 2: Efficient checking of large model

```
1-  Input: test dataset, M: a large model, t: a given reachability property to be checked;
2-  Output: a witness for verification property reachability;
3-  Path = the first row of the test dataset;
4-  while Path <> null do
5-      RF= Random forest (path);
6-      if  RF.predict (path) equals 1 then
7-          the path as a promising path and apply the path on the M;
8-      if  current state is a goal state then
9-          return starting from the initial state and leading to the current state as a witness;
10-     end if;
11-     end if;
12-    path =next row;
13- end while;
```

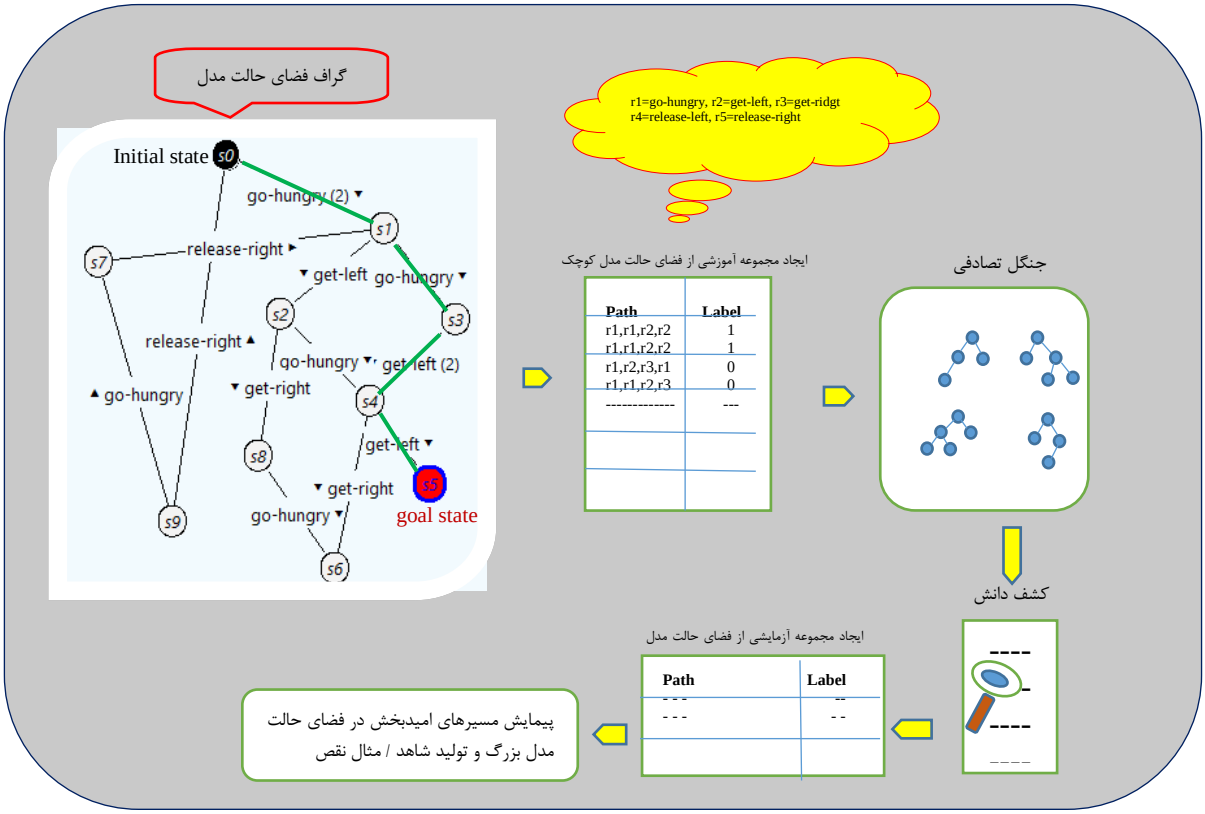
در الگوریتم (۲) ابتدا هر سطر مجموعه آزمایشی را یک مسیر (*path*) در نظر می‌گیریم. دانش کشف‌شده از فضای حالت مدلِ کوچک (مجموعه آموزشی) را بر روی مسیر جاری اعمال می‌کنیم. چنانچه روابط منطقی موجود در مسیر جاری منطبق بر دانش کشف‌شده باشد، نتیجه پیش‌بینی برابر یک می‌شود. در این صورت، مسیر موجود به‌عنوان یک مسیر امیدبخش در نظر گرفته‌شده و بر روی مدل M (مدل بزرگ) پیمایش می‌شود. چنانچه در انتهای مسیر به یک حالت هدف رسیدیم، با شروع از حالت ابتدایی تا انتهای مسیر پیمایش‌شده به‌عنوان یک شاهد (*witness*) برای تأیید ویژگی دسترس‌پذیری حالت t نشان داده می‌شود. در غیر این صورت روند بالا برای سطر بعدی در مجموعه آزمایشی تکرار می‌شود.

۳-۳-۳- مثال جامع

در اینجا با بیان یک مثال جامع عملکرد روش پیشنهادی جهت تأیید ویژگی دسترس‌پذیری حالت t، به‌طور دقیق بیان می‌شود. مسأله غذاخوردن فیلسوفان را که با زبان رسمی تبدیل گراف، مدل‌سازی شده است، در نظر می‌گیریم. فرض کنید حالت t برابر است با « همه فیلسوفان چنگال سمت چپ را برداشته باشند و منتظر چنگال سمت راست هستند.»

[DOI: 10.52547/jsdp.19.1.153]

در ابزار GROOVE از الگوریتم جستجوی BFS برای پیمایش فضای حالت در مدل‌های با اندازه حداکثر دوازده فیلسوف استفاده می‌شود؛ اما این ابزار در تولید فضای حالت مدلی با بیشتر از این تعداد فیلسوف ناتوان بوده و با مشکل انفجار فضای حالت مواجه می‌شود. برای حل این مشکل روش پیشنهادی می‌تواند کارآمد باشد. برای مثال بخواهیم دسترس‌پذیری حالت t را در مدل بزرگ با اندازه بیست فیلسوف وارسی کنیم. مطابق با روش پیشنهادی، ابتدا مدل کوچک‌تر به‌عنوان‌مثال با دو فیلسوف را ایجاد می‌کنیم. تمام مسیرهای گرافی که از حالت اولیه S_0 شروع و به آخرین سطح ختم می‌شوند پس از برچسب‌گذاری، در مجموعه آموزشی ذخیره می‌شوند (مسیرهایی که به حالت هدف ختم می‌شوند با مقدار یک و در غیر این صورت با مقدار صفر برچسب‌گذاری می‌شود). مجموعه آموزشی به الگوریتم جنگل تصادفی آموزش داده می‌شود تا دانش حاکم بر فضای حالت مدل کوچک کسب شود. در ادامه مسیرهای موجود گراف، فضای حالت مدل بزرگ را تا سطح مشخصی (تا جایی که حافظه ظرفیت داشته باشد) استخراج و در مجموعه آزمایشی قرار می‌دهیم. مجموعه آزمایشی در اختیار جنگل تصادفی قرار داده می‌شود تا به توجه به دانش کسب‌شده، برچسب مسیرهای را پیش‌بینی و تعیین کند. مسیرهایی که برچسب یک دریافت می‌کنند، به‌عنوان مسیرهای امیدبخش در نظر گرفته می‌شود. درنهایت فقط مسیرهای امیدبخش در مدل بزرگ برای یافتن حالت هدف پیمایش می‌شوند. در صورت یافتن حالت هدف، یک شاهد تولید می‌شود. شکل (۱) جزئیات روش پیشنهادی را برای این مثال نشان می‌دهد.



(شکل-۱): نمودار جعبه‌ای مثال جامع
(Figure-1): Diagram of the comprehensive example

۴- مفاهیم اولیه

۴-۱- وارسی مدل

وارسی مدل یک روش کارآمد و خودکار جهت درستی‌یابی سامانه‌های نرم‌افزاری در مرحله تحلیل و طراحی است. مطالعات نشان می‌دهند که خطایابی و هزینه رفع آن‌ها در مرحله تحلیل و طراحی نسبت به مراحل آزمون و نگهداری، به‌طور چشم‌گیری کمتر است. در فرایند وارسی مدل، ابتدا مدلی از سامانه ایجادشده، سپس به همراه ویژگی موردنظر به یک وارسی‌کننده مدل وارد می‌شود. وارسی‌کننده مدل به‌صورت نظام‌مند و خودکار با کاوش در تمام فضای حالت مدل، درستی یا نادرستی ویژگی را بررسی می‌کند. ویژگی که مورد صحت سنجی (درستی آزمایی) قرار می‌گیرد اغلب به‌صورت یک منطق زمانی

[DOI: 10.52547/jsdp.19.1.153]

مانند منطق خطی (LTL) یا منطق درخت محاسباتی (CTL) توصیف می‌شود. در صورت موفقیت‌آمیز بودن عملیات وارسی مدل (مشکل انفجار فضای حالت رخ ندهد)، یک مثال نقص / شاهد توسط وارسی‌کننده مدل، تولید می‌شود. مثال نقص‌ها / شاهدها می‌توانند توسط متخصصان جهت اصلاح معایب طراحی مورداستفاده قرار گیرند. یک مثال نقص / شاهد میسری در فضای حالت سامانه است که از حالت ابتدایی شروع و به حالت هدف ختم می‌شود. حالت هدف، حالتی است که در آن ویژگی موردنظر انکار/ تأیید می‌شود. از ویژگی‌های مهمِ سامانه‌های نرم‌افزاری که توسط این روش مورد وارسی قرار می‌گیرند، می‌توان به موارد ایمنی و دسترس‌پذیری اشاره کرد. ازآنجاکه تأیید ویژگی ایمنی در سامانه‌های بزرگ و پیچیده مستلزم بررسی تمام حالت‌های قابل‌دسترس است، ممکن است انفجار فضای حالت رخ دهد. در چنین سامانه‌هایی تلاش می‌شود تا ویژگی ایمنی اِنکار شود. به این مفهوم که دسترس‌پذیری یک حالت خطا (مانند وجود بن‌بست یا یک رفتار نامطلوب) در فضای حالت سامانه مورد کاوش قرار می‌گیرد. چنانچه دسترس‌پذیری این حالت خطا تأیید شود، می‌توان ویژگی ایمنی در سامانه را رد کرد.

۲-۴- سامانه تبدیل گراف

سامانه تبدیل گراف ابزار مناسبی برای توصیف و مدل‌کردن سامانه است. گراف‌ها در کنار سادگی و بصری‌بودن، دارای پایه ریاضی و رسمی دقیق هستند. در سامانه تبدیل گراف اجزای سامانه به شکل رأس‌های گراف و روابط بین آن‌ها به‌صورت یال‌های گراف ترسیم می‌شوند. یک سامانه تبدیل گراف به‌صورت سه تایی *AGT(TG,HG,R)* توصیف می‌شود [30]. *TG* گراف نوع بوده و اجزای سامانه را به همراه نام و مجموعه صفاتی که هر یک اجزا می‌گیرند، نشان می‌دهد. HG گراف میزبان است و پیکربندی اولیه سامانه را نشان می‌دهد. گراف میزبان HG گرافی هم‌ریخت با گراف نوع TG است. به‌طوری‌که هر گره و یال در گراف میزبان به یک گره و یال در گراف نوع نگاشت می‌شود. *R* مجموعه قوانین تبدیل گراف است. از یک نام به همراه دو گراف میزبان LHG و RHG تشکیل‌شده و به‌صورت R:LHS → RHG نشان داده می‌شود [31]. با اعمال هر قانون *R*، سامانه از یک حالت به حالت دیگر تغییر می‌کند.

۳-۴- روش جنگل تصادفی

یکی از راه‌کارهای موجود جهت بهبود صحت مدل، استفاده ترکیبی از چند مدل به‌جای تنها یک مدل است. الگوریتم‌هایی ترکیبی، الگوریتم‌هایی هستند که یک مجموعه از مدل‌ها را گرفته و خروجی آن‌ها را با یکدیگر ترکیب می‌کنند تا دسته‌بندی نهایی را به‌گونه‌ای بسازند که کارایی آن از کارایی تک‌تک دسته‌بندهای پایه استفاده‌شده در الگوریتم بیشتر باشد. درنهایت برچسب ردهٔ رکوردهای جدید، با ترکیب‌کردن خروجی تک‌تک مدل‌های پایه‌ی استفاده‌شده تعیین می‌شود. در این مقاله از روش جنگل تصادفی استفاده شده است. دسته‌بندهای استفاده‌شده در جنگل تصادفی همگی از نوع درخت تصمیم هستند. روند کلی برای تولید T درخت تصمیم بدین قرار است. در هر تکرار (t=1,2,..,T) با روش نمونه‌گیری با جایگزینی، مجموعه آموزشی Di درست می‌شود. ازآنجایی‌که از روش نمونه‌گیری با جایگزینی استفاده می‌شود، ممکن است برخی از تاپل‌ها بیشتر از یک‌بار در Di حضورداشته باشند و برخی دیگر نیز در این مجموعه آموزشی قرار نگیرند. تعداد صفات خاصه‌ای که برای تعیین انشعاب در هر گره درخت استفاده می‌شوند با m نشان داده شده است. این تعداد صفات خاصه کمتر از تعداد صفات خاصه موجود و در دسترس است. از میان m صفت خاصه که در هر گره نامزد انشعاب هستند، صفت خاصه‌ای که دارای بالاترین میزان بهره است به‌عنوان صفت خاصه انشعاب، انتخاب می‌شود. الگوریتم (۳) شبه‌کد روش یادگیری جنگل تصادفی را نشان می‌دهد.

Algorithm 3: Random forest- to create a composite model of classifiers.

```
1-  Input: D, a set of 2 class- label training dataset.
2-  T, the number of trees.
3-  B, the number of nodes.
4-  X: new Tuple
5-  Output: A composite model.
6-  for t=1: T do
7-        Randomly sample the train data D with replacement to produce Di
8-        Grow an unpruned decision tree.
9-  for b=1 to B do
10-       Select m variable at random from the ρ variables.
```


[DOI: 10.52547/jsdp.19.1.153]

```
11-          Pick the best variable with the highest information gain among the m.
12-          Split the node into two daughter nodes.
13-     end for
14-     end for
15-   To make a prediction at a new Tuple X:
16-          return majority vote the predictions of the T trees.
```

۵- پیاده‌سازی و نتایج تجربی

در این بخش کارایی روش پیشنهادی برای تأیید ویژگی دسترس‌پذیری مورد ارزیابی قرار می‌گیرد. برای ارزیابی راه‌حل ارائه‌شده و مقایسه کارایی آن با روش‌های دیگر، آن را با زبان برنامه‌نویسی جاوا و در ابزار متن‌باز GROOVE پیاده‌سازی کرده‌ایم. روش پیشنهادی با روش‌های مبتنی بر جستجوی مکاشفه‌ای مانند: BS[32]، IDA*[33]، BFS، DFS، روش‌های فرامکاشفه‌ای و تکاملی مانند: GA[26]، PSO[34]، PSO-GSA، روش مبتنی بر بهینه‌سازی بیزین (BOAtcp) [27] و روش مبتنی بر یادگیری ماشین (model checking + Adaboost) [28] مورد مقایسه و ارزیابی قرارگرفته است. آزمایش‌ها با استفاده از پردازنده Intel® Core™ i5، حافظه 3GB و سیستم‌عامل Windows 8 Ultimate انجام‌شده است.

۵-۱- مدل مسائل استفاده‌شده

روش پیشنهادی بر روی چهار مسأله معروف که وارسی آن‌ها در سامانه‌های تبدیل گراف، امکان‌پذیر نیست، مورد آزمایش و وارسی قرارگرفته است. این مسائل عبارت‌اند از: غذاخوردن فیلسوفان [29]، خوانندگان-نویسندگان [30]، N- وزیر و چرخه حیات پردازه [31].

۵-۱-۱- مسأله ناهارخوردن فیلسوف‌ها

در این مسأله، n فیلسوف به همراه n چنگال بین آن‌ها دور یک میز نشسته‌اند. هر فیلسوف فکر می‌کند، سپس گرسنه می‌شود و برای خوردن غذا، ابتدا چنگال چپ، سپس چنگال راست را برمی‌دارد و شروع به خوردن غذا می‌کند. ازآنجایی‌که هر چنگال بین دو فیلسوف مجاور مشترک است، بنابراین فیلسوفان بر سر برداشتن چنگال با هم رقابت می‌کنند.

۵-۱-۲- مسأله خوانندگان-نویسندگان

در این مسأله چندین پردازه برای دسترسی هم‌زمان به منابع مشترک با هم رقابت می‌کنند. بعضی از پردازنده‌ها قصد خواندن منابع دارند و در نقش خواننده هستند و برخی دیگر در نقش نویسنده نیاز به نوشتن در منابع دارند. قاعده این است که چندین خواننده مجازند به‌طور هم‌زمان از یک منبع بخوانند به شرطی که هیچ نویسنده‌ای در حال نوشتن آن نباشد. همچنین در هرلحظه فقط یک نویسنده می‌تواند در یک منبع بنویسد.

۵-۱-۳- مسأله N – وزیر

این مسأله شامل N وزیر در یک صفحه شطرنج $N \times N$ است. وزیرها باید طوری در صفحه چیده شوند که نتوانند همدیگر را گارد دهند. هدف از این مسأله، یافتن یک چیدمان از این وزیرها است، به‌طوری‌که هیچ‌کدام نتواند دیگری را گارد دهد.

۵-۱-۴- مسأله چرخه حیات پردازه‌ها

در این مسأله چرخه زندگی پردازه توصیف می‌شود. در چرخه حیات، بعد از ایجاد پردازه، در صورتِ وجود حافظه کافی، به حافظه بارگذاری شده و منتظر *CPU* یا وسایل *I/O* می‌ماند. بعد از کامل‌شدن اجرا، پردازه منابع در اختیارش را آزاد می‌کند و متوقف می‌شود.

۵-۲- نتایج و ارزیابی

نتایج اجرای هر مسأله، در دو جدول مجزا آورده شده است. در نخستین جدول، متوسط زمان اجرای روش پیشنهادی به همراه متوسط زمان اجرای دیگر رویکردها نشان داده شده است. دومین جدول، جزئیات اجرای روش پیشنهادی را نشان می‌دهد که شامل تعداد حالت‌های پیمایش‌شده، نخستین عمق دسترسی به حالت هدف، کمینه و بیشینه زمان اجرای روش پیشنهادی گزارش‌شده است. در انتهای این بخش، طول شاهد تولیدشده به‌وسیلهٔ هرکدام از روش‌ها در جدول (۹) آورده شده است.

اگرچه الگوریتم‌های *BFS* و *DFS* در ابزار *GROOVE* پیاده‌سازی شده‌اند و از جستجوی کامل فضای

[DOI: 10.52547/jsdp.19.1.153]

حالت برای یافتن حالت هدف استفاده می‌کنند، اما در مدل‌های بزرگ با مشکل انفجار فضای حالت روبه‌رو می‌شوند و در هیچ از مسائل مطرح‌شده، نمی‌توانند حالت هدف را شناسایی کنند. درضمن، اگر روشی بنا بر دلایلی از جمله اتمامِ همه حافظهٔ در دسترس یا تعداد تکرارهای مُجاز نتواند هیچ حالت هدفی را در مدل مفروض پیدا کند، عبارت Not found را در ستون مربوطه نشان خواهیم داد.

در مسأله ناهارخوردن فیلسوفان، از مدل کوچک با سه فیلسوف برای یادگیری و کشف فضای حالت استفاده شده است. در این مسأله، ویژگیِ حالتِ q به‌صورت: تمام فیلسوفان چنگال سمت چپ را برداشته و منتظر چنگال سمت راست هستند، در نظر گرفته می‌شود. جدول (۱) نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری حالت q را نشان می‌دهد. الگوریتم *IDA فضای حالت را به‌صورت عمقی پیمایش می‌کند و ممکن است حالت هدف تا پیمایش کامل فضای حالت پیدا نشود؛ بنابراین در مدل‌های بزرگ، با توجه به حجیم‌بودن فضای حالت این احتمال وجود دارد که حالت هدف به‌وسیلهٔ این الگوریتم پیدا نشود. الگوریتم BS برخلاف الگوریتم *IDA فضای حالت را به‌صورت عمقی و سطحی پیمایش می‌کند و بدین ترتیب این الگوریتم شانس بالایی برای یافتن حالت هدف دارد؛ ولی با وجود این نیز، این الگوریتم در مدل‌های بزرگ با شکست مواجه می‌شود. مطابق اطلاعات جدول (۱)، روش پیشنهادی در مدل‌های بزرگ‌تر عملکرد مناسب‌تری نسبت به روش‌های دیگر دارد. جزئیات روش پیشنهادی در جدول (۲) گزارش شده است.

(جدول-۱): مقایسه نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسأله ناهار خوردن فیلسوفان
(Table-1): Comparison of running times of all approaches to solving the dining philosophers problem

تعداد فیلسوف‌ها	Depth limit	GA(Sec)	PSO(Sec)	PSO-GSA(Sec)	BS(Sec)	IDA*(Sec)	Model checking+ AdaBoost (Sec)	BOActp (Sec)	روش پیشنهادی (Sec)
۲۰	۱۰۰	۱۲,۳۸	۶۲,۳۲	۵۵,۲۸	۱۱۶,۶۳	Not found	۵,۷	۸,۹۵	۵,۵۲
۲۵	۱۵۰	۲۷,۳۱	۸۷,۶۴	۸۲,۸۴	۳۴۲,۵۴		۶,۷۸	۲۰,۷۸	۶,۱۷
۳۰	۲۰۰	۷۵,۶۲	۱۲۳,۷۹	۱۰۲,۹۴	۸۷۳,۲۳		۷,۰۸	۴۱,۸۲	۶,۹۶

(جدول-۲): جزئیات اجرای روش پیشنهادی در مسأله‌ی ناهار خوردن فیلسوفان
(Table-2): Details of running the proposed approach to the dining philosophers problem

تعداد فیلسوف‌ها	تعداد حالت‌های پیمایش شده	اولین عمق دسترسی به حالت هدف	حداکثر زمان اجرا (Sec)	حداقل زمان اجرا (Sec)
۲۰	۱۳۶۲	۴۰	۶,۲۱	۵,۰۳
۲۵	۲۲۹۰	۵۰	۶,۷۳	۵,۳۴
۳۰	۳۸۵۳	۶۰	۷,۰۱	۵,۹۲

در مسأله خوانندگان- نویسندگان، از دو خواننده و دو نویسنده برای ایجاد مدل کوچک استفاده شده است. با فرض این‌که ویژگی حالت q در مسأله خوانندگان-نویسندگان به‌صورت: همه خوانندگان–نویسندگان پردازش‌هایشان را به پایان رسانده باشند» فضای حالت این مسأله در مکان‌های عمیق، پهن و گسترده است. به همین خاطر زمان اجرای روش BS در مدل‌های بزرگ زیاد خواهد شد. روش پیشنهادی در مدل‌های بزرگ‌تر دارای کارایی بهتری نسبت به بقیه روش‌ها است. جدول (۳) نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری حالت q را نشان می‌دهد. جزئیات روش پیشنهادی در جدول (۴) گزارش‌شده است.

[DOI: 10.52547/jsdp.19.1.153]

(جدول-۳): مقایسه نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسألهٔ خوانندگان- نویسندگان

(Table-3): Comparison of running times of all approaches to solving the readers-writers problem

تعداد خوانندگان – نویسندگان	Depth limit	GA(Sec)	PSO(Sec)	PSO-GSA(Sec)	BS(Sec)	IDA*(Sec)	Model checking+ AdaBoost (Sec)	BOActp (Sec)	روش پیشنهادی (Sec)
4-R-4-W	۵۰	۹،۷۲	۹،۲	۱۳	۲۵۴	۴	۶،۰۵	۱،۹۸	۴
5-R-5-W	۷۰	۶۳،۳	۳۸	۱۹	۴۹۲	۵	۸،۳	۷،۱۵	۵
6-R-6-W	۹۰	۱۶۴	۷۴	۵۳	۵۴۹	۷	۱۰،۷۶	۳۵،۶۵	۶،۴۳

(جدول-۴): جزئیات اجرای روش پیشنهادی در مسألهٔ خوانندگان-نویسندگان

(Table-4): Details of running the proposed approach to the readers-writers problem

تعداد خوانندگان- نویسندگان	تعداد حالت‌های پیمایش شده	اولین عمق دسترسی به حالت هدف	حداکثر زمان اجرا (Sec)	حداقل زمان اجرا (Sec)
4-R-4W	۱۰۶۳	۲۵	۴،۵	۳،۲۹
5-R-5-W	۱۳۳۹	۴۸	۵،۶۲	۴،۱۱
6-R-6-W	۱۷۷۴	۶۵	۷،۱۹	۵،۹۵

در مسأله N وزیر، مدل کوچک با ابعاد ۴×۴ ایجاد شده است. در این مسأله ویژگی حالت q به‌صورت: همه وزیرها در موقعیت صحیح قرارگرفته باشند در نظر گرفته شده است. نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری حالت q در جدول (۵) آورده شده است. اگرچه در این مسأله تنها دو قانون وجود دارد، ولی دارای فضای حالت بسیار پهن و گسترده است؛ بنابراین الگوریتم BS نمی‌توانند هیچ حالت هدفی را پیدا کنند. الگوریتم IDA* می‌تواند حالت هدف را در مدل‌های کوچک ۸×۸ پیدا کند؛ اما در مدل‌های بزرگ‌تر قادر به شناسایی هیچ حالت هدفی نیست. راه‌حل پیشنهادی به‌ویژه در مدل‌های بزرگ‌تر زمان اجرای کوتاه‌تری دارد. جزئیات اجرای روش پیشنهادی در جدول (۶) گزارش شده است.

(جدول-۵): مقایسه نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسألهٔ N- وزیر

(Table-5): Comparison of running times of all approaches to solving the N-Queen problem

ابعاد	Depth limit	GA(Sec)	PSO(Sec)	PSO-GSA(Sec)	IDA*(Sec)	BS(Sec)	Model checking+ AdaBoost (Sec)	BOActp (Sec)	روش پیشنهادی (Sec)
۸×۸	۱۰۰	۶،۳۱	۲۸،۱۹	۳۱،۹۴	۱۱۷،۹۳	Not found	۱۰،۷۳	۳،۷	۱۰،۵۱
۱۶×۱۶	۱۰۰	Not found					۲۷،۳	Not found	۲۳،۵۴
۲۰×۲۰	۱۰۰						۵۲،۴۱		۴۲،۱۵

(جدول-۶): جزئیات اجرای روش پیشنهادی در مسألهٔ N- وزیر

(Table-6): Details of running the proposed approach to the N-Queen problem

ابعاد	تعداد حالت‌های پیمایش شده	اولین عمق دسترسی به حالت هدف	حداکثر زمان اجرا (Sec)	حداقل زمان اجرا (Sec)
۸×۸	۲۷۹۱	۱۰	۱۱/۹۶	۹/۴۲
۱۶×۱۶	۴۹۰۷	۲۲	۲۷/۵	۲۱/۷۳
۲۰×۲۰	۶۷۴۳	۲۴	۴۸/۰۴	۳۹/۲۷

در مسأله چرخه حیات پردازه، مدل کوچک با سه پردازه و سه حافظه ایجاد شده است. ویژگی حالت q به‌صورت «همه پردازش‌ها اجراهایشان را تمام کرده باشند»، فرض شده است. جدول (۷)، زمان اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری حالت q را نشان می‌دهد. در این مسأله، حالت‌های هدف در مکان‌های عمیق فضای حالت قرار دارد. به‌جز روش پیشنهادی و الگوریتم‌های IDA*، BOActp و Model checking+AdaBoost بقیه روش‌ها در مدل‌های بیشتر از دوازده پردازه قادر به یافتن حالت هدف نیستند. الگوریتم

[DOI: 10.52547/jsdp.19.1.153]

BS نیز قبل از رسیدن به حالت هدف با کمبود حافظه مواجه می‌شود و درنتیجه دررسیدن به حالت هدف ناموفق است. جزئیات روش پیشنهادی در جدول (۸) گزارش شده است.

(جدول-۷): مقایسه نتایج اجرای همه روش‌ها برای تأیید ویژگی دسترس‌پذیری در مسألهٔ N- وزیر
(Table-7): Comparison of running times of all approaches to solving the N-Queen problem

ابعاد	Depth limit	GA(Sec)	PSO(Sec)	PSO-GSA(Sec)	IDA*(Sec)	BS(Sec)	Model checking+ AdaBoost (Sec)	BOActp (Sec)	روش پیشنهادی (Sec)
۸×۸	۱۰۰	۶،۳۱	۲۸،۱۹	۳۱،۹۴	۱۱۷،۹۳	Not found	۱۰،۷۳	۳،۷	۱۰،۵۱
۱۶×۱۶	۱۰۰	Not found					۲۷،۳	Not found	۲۳،۵۴
۲۰×۲۰	۱۰۰						۵۲،۴۱		۴۲،۱۵

(جدول-۸): جزئیات اجرای روش پیشنهادی در مسألهٔ N- وزیر
(Table-8): Details of running the proposed approach to the N-Queen problem

ابعاد	تعداد حالت‌های پیمایش شده	اولین عمق دسترسی به حالت هدف	حداکثر زمان اجرا (Sec)	حداقل زمان اجرا (Sec)
۸×۸	۲۷۹۱	۱۰	۱۱،۹۶	۹،۴۲
۱۶×۱۶	۴۹۰۷	۲۲	۲۷،۵	۲۱،۷۳
۲۰×۲۰	۶۷۴۳	۲۴	۴۸،۰۴	۳۹،۲۷

در این مقاله طول شاهد تولیدشده را به‌عنوان معیار ارزیابی دوم برای سنجش کارایی روش پیشنهادی با روش‌های قبلی در نظر گرفته شده است. برای مقایسه طول شاهدان تولیدشده به‌وسیلهٔ رویکردها، ما یک نمونه از هر مسأله را انتخاب کرده‌ایم. جدول (۹) نتایج را نشان می‌دهد. همان‌طور که در جدول (۹) مشاهده می‌شود در بیش‌تر مدل‌ها روش پیشنهادی موفق به تولید شاهد کوتاه‌تر شده است. گفتنی است که میانگین نتایج مربوط به روش پیشنهادی بر اساس بیست بار اجرای مستقل و موفق گزارش شده است.

(جدول-۹): مقایسه طول تولید شاهد توسط روش‌های مختلف در نمونه مدلی از مسائل مختلف
(Table-9): Comparison of the length of the generated witnesses by all approaches

مدل‌ها / روش‌ها	Depth limit	GA	PSO	PSO-GSA	IDA*	BS	Model checking+ AdaBoost	BOActp	روش
Dining philosophers (15 philosophers)	۵۰	۴۸،۵۳	۵۶،۰۳	۴۶،۱۲	Not found	۳۰	۳۲،۰۸	۴۹،۶۸	۳۲،۷۳
Readers-writers (4-R-4-W)	۵۰	۴۹،۸۴	۴۸،۱۷	۴۸،۱۳	۴۸	۳۲	۳۰،۱۷	۵۰،۷۸	۲۶،۴۱
Process life cycle (۱۰ پردازه- ۱۰ حافظه)	۴۰	۳۸،۲۲	۳۹،۳۶	۳۸،۲	۳۰	Not found	۱۷،۵۴	۴۱،۳۹	۱۱،۰۹
N-Queen (8×8)	۱۰۰	۶۱،۱۸	۵۶،۷۲	۵۲،۳۱	۹۸		۴۸،۷۲	۷۴،۵۲	۳۸،۸۵

۶- نتیجه‌گیری

الگوریتم جنگل تصادفی دقت بسیار بالایی شبیه به الگوریتم‌های *SVM* و شبکه عصبی دارد؛ اما آموزشِ جنگل تصادفی نسبت به آن‌ها سریع‌تر است. جنگل تصادفی بر روی مجموعه‌های بزرگ عملکرد خوبی دارد و به‌کارگیری آن نیاز به پارامترهای کمتری دارد؛ بنابراین هوشمندی و سرعت عملیات وارسی مدل را افزایش می‌دهد. روش پیشنهادی در مقایسه با روش‌های دیگر، سرعت اجرای بالاتری دارد و به‌طور هوشمندانه‌تری فضای حالت مدل را پیمایش و شاهد کوتاه‌تری تولید می‌کند.

روش پیشنهادی دارای محدودیت‌هایی نیز است. نخست این‌که، باید مدل کوچک مناسبی از سامانه ایجاد شود. طراحی چنین مدلی در برخی موارد سخت و حتی

[DOI: 10.52547/jsdp.19.1.153]

غیرممکن است. دوم اینکه کارایی روش پیشنهادی به میزان و کیفیت دانش کسب‌شده از فضای حالت مدل کوچک به‌وسیلهٔ الگوریتم یادگیری ماشین وابسته است.

برای رفع محدودیت‌های ذکرشده پیشنهاد می‌شود در کارهای آینده به‌جای تولید مدل کوچک مسأله، بخشی از فضای حالت مدل بزرگ مسأله تولید و به‌منظور یادگیری و کشف دانش به الگوریتم جنگل تصادفی داده شود. همچنین می‌توان از الگوریتم‌های دیگری مانند روش یادگیری بیزین و یادگیری عمیق نیز استفاده کرد.

۷- مراجع 7- References

[1] H. Zhang, J. Du, L. Cao and G. Zhu, "A full symbolic reachability analysis algorithm of timed automata based on BDD", *In Autonomous Decentralized Systems (ISADS)*, IEEE Twelfth International Symposium, pp. 301-304, 2015.

[2] A. L. Lafuente, "Symmetry reduction and heuristic search for error detection in model checking", Workshop on Model Checking and Artificial Intelligence, 2003.

[3] A. L. Lafuente, S. Edelkamp, and S. Leue, "Partial order reduction in directed model checking", *International SPIN Workshop on Model Checking of Software*, Springer Berlin Heidelberg, pp. 112-127, 2002.

[4] V. Rafe, "Scenario-driven analysis of systems specified through graph transformations", *Journal of Visual Languages & Computing*, vol. 24, no. 2, pp. 136-145, 2013.

[5] A. Rensink and E. Zambon, "Pattern-based graph abstraction in Graph transformations", *Springer Berlin Heidelberg*, pp 66-80, 2012.

[6] S. Edelkamp, A. L. Lafuente, and S. Leue, "Protocol verification with heuristic search", AAAI Symposium on Model based Validation of Intelligence, 2001.

[7] S. Edelkamp, A. L. Lafuente, and S. Leue, "Directed explicit model checking with HSF-SPIN", *Proceedings of the 8th international SPIN workshop on Model checking of software*, pp. 57-79,2001.

[8] S. Edelkamp and F. Reffel, "OBDDs in heuristic search", *Annual Conference on Artificial Intelligence*, Springer Berlin Heidelberg, pp. 81-92, 1998.

[9] J. Maeoka, Y. Tanabe, and F. Ishikawa, "Depth-First Heuristic Search for Software Model Checking", *In Computer and Information Science, Springer International Publishing*, pp. 75-96, 2016.

[10] K. Havelund, T. Pressburger, "Model checking JAVA programs using JAVA Path Finder", *International Journal on Software Tools for Technology Transfer*, vol. 2, pp. 366–381,2000

[11] H. C. Estler and H. Wehrheim, "Heuristic search-based planning for graph transformation systems", *KEPS 2011*, pp. 54-66, 2011.

[12] E. Snippe, "Using heuristic search to solve planning problems in GROOVE", In 14th Twente Student Conference on IT, University of Twente, 2011.

[13] S. Ziegert, "Graph Transformation Planning via Abstraction", arXiv preprint arXiv: 1407.7933. 2014.

[14] J. W. Elsinga, "On a framework for domain independent heuristics in graph transformation planning", Master's thesis, University of Twente, 2016.

[15] P. Godefroid and S. Khurshid, "Exploring very large state spaces using genetic algorithms", *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, pp. 266-280, 2002.

[16] P. Godefroid, "Software Model Checking: The VeriSoft Approach Patrice Godefroid, Formal Methods in System Design". vol.26, pp. 77–101, 2005

[17] R. Yousefian, V. Rafe, M. Rahmani, "a heuristic approach for model checking graph transformation systems", *Appl. Soft Compute*, Vol. 24, pp. 169–180, 2014

[18] X. He, Z. Ma, W. Shao, G. Li, "A meta model for the notation of graphical modeling languages", *In Computer Software and Applications Conference*, COMPSAC 2007. 31st Annual International, IEEE, vol. 1, pp. 219-224, 2007.

[19] GROOVE, *groove.sourceforge.net/groove-index.html*

[20] R. Yousefian, S. Aboutorabi, and V. Rafe, "A greedy algorithm versus meta heuristic solutions to deadlock detection in Graph Transformation Systems", *Journal of Intelligent & Fuzzy Systems*, vol. 13, no. 1, pp. 1-13, 2016.

[21] E. Alba, F. Chicano, M. Ferreira, and J. Gomez-Pulido, "finding deadlocks in large concurrent java programs using genetic algorithms", *10th annual conference on Genetic and evolutionary computation*, pp. 1735-1742, 2008.

[22] L. M. Duarte, L. Foss, R. Wagner, and T. Heimfarth, "Model Checking the Ant Colony Optimization, Distributed, Parallel and Biologically Inspired Systems IFIP Advances", *Information and Communication Technology*, vol. 329, pp. 221-232, 2010

[23] B. L. Webster, "solving combinatorial optimization problems using a new algorithm

[DOI: 10.52547/jsdp.19.1.153]

based on gravitational attraction", Florida Institute of Technology, 2004.

[24] R. Behjati, M. Sirjani, and M. N. Ahmadabadi, "Bounded Rational Search for On-the-Fly Model Checking of LTL Properties", *Fundamentals of Software Engineering*, vol. 5961, pp. 292-307, 2010

[25] G.J. Holzmann, "*On-The-Fly Model Checking*", ACM Comput Surv, vol.28(4es), pp.120, 1996

[26] E. Pira, V. Rafe, A. Nikanjam, "*EMCDM: efficient model checking by data mining for verification of complex software systems specified through architectural styles*", Appl. Soft Compute, Vol. 44pp, pp.1185–1201, 2016.

[27] E. Pira, V. Rafe, A. Nikanjam, "Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm", *Journal of System and Software*, vol. 131, pp. 181-200, 2017

[28] J. Partabian, V. Rafe, H. Parvin, S. Nejatian, "An Approach Based on Knowledge Exploration for State Space Management in Checking Reachability of Complex Software Systems", *soft computing*, vol. 24, pp.1-16, 2019.

[29] M. Yasrebi, V. Rafe, H. Parvin, S. Nejatian, "An efficient approach to state space management in model checking of complex software system using machine learning technique", *journal of intelligent & Fuzzy system*, vol.38, no. 2, pp. 1761-1773, 2020.

[30] G. Rozenberg, "Handbook of Graph Grammars and Comp", *World scientific*, vol. 1, 1997.

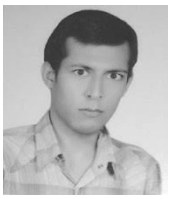
[31] H. Kastenberg and A. Rensink, "Model Checking Dynamic States in GROOVE", *International SPIN Workshop on Model Checking of Software*, Springer Berlin Heidelberg, pp. 299-305, 2006.

[32] A. Groce and W. Visser, "Heuristics for model checking Java programs", *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 6, no. 4, pp. 260-276, 2004.

[33] S. Edelkamp, A. L. Lafuente, and S. Leue, "Protocol verification with heuristic search", *In AAAI Symposium on Model-based Validation of Intelligence*, pp. 75-83, 2001.

[34] V. Rafe, M. Moradi, R. Yousefian, and A. Nikanjam, "A Meta-Heuristic Approach for Automated Refutation of Complex Software Systems Specified through Graph Transformations", *Applied Soft Computing*, vol. 33, pp. 136-149, 2015.

وحید رافع کارشناسی، کارشناسی ارشد و دکترای خود را در رشته مهندسی رایانه از دانشگاه علم و صنعت گذرانده‌اند. فرصت مطالعاتی شش‌ماهه در دانشگاه پلی‌روش میلان داشته‌اند و در حال حاضر دانشیار دانشگاه اراک هستند. زمینه‌های مطالعاتی موردعلاقه ایشان، تبدیل مدل، تجمع و آزمایش نرم‌افزار و روش‌های رسمی است.
نشانی رایانامه ایشان عبارت است از:
v-rafe@araku.ac.ir



حمید پروین کارشناسی در دانشگاه اهواز، کارشناسی ارشد و دکترای خود را در رشته مهندسی رایانه دانشگاه علم و صنعت گذرانده‌اند. هم‌اکنون عضو هیئت‌علمی دانشگاه آزاد واحد نورآباد ممسنی هستند.
نشانی رایانامه ایشان عبارت است از:
parvinhamid@gmail.com

صمد نجاتیان دکترای خود را از دانشگاه UTM مالزی دریافت کرد. هم‌اکنون عضو هیئت‌علمی دانشگاه آزاد واحد یاسوج است.
نشانی رایانامه ایشان عبارت است از:
samad.nej.2007@gmail.com

جعفر پرتابیان کارشناسی ارشد خود را در رشته مهندسی رایانامه در دانشگاه علوم و تحقیقات گذرانده و در حال حاضر دانشجوی دکترای دانشگاه آزاد اسلامی واحد یاسوج است.
نشانی رایانامه ایشان عبارت است از:
Jaafar_partabian@yahoo.com

کرم‌الله باقری‌فرد مدرک کارشناسی خود را در سال ۱۳۸۴ در رشته مهندسی کامپیوتر گرایش نرم‌افزار از دانشگاه اصفهان و مدرک کارشناسی ارشد و دکترای خود را به‌ترتیب در سال‌های ۱۳۸۷ و ۱۳۹۵ از دانشگاه نجف‌آباد و اراک در رشته مهندسی کامپیوتر گرایش نرم‌افزار اخذ کرد. وی از سال ۱۳۸۵ تاکنون عضو هیأت علمی بخش مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد یاسوج است. حوزه‌های تخصصی ایشان داده‌کاوی، یادگیری ماشین و سیستم‌های پیشنهاددهنده است. وی تاکنون بیش از ۸۰ مقاله علمی در نشریات و کنفرانس‌های معتبر داخلی و خارجی به چاپ رسانیده است.
نشانی رایانامه ایشان عبارت است از:
k.bagheri@iauyasooj.ac.ir

[DOI: 10.52547/jsdp.19.1.153]