Archive of SID

مقاله پژوهشی

یک هیوریستیک جدید برای تشخیص بن‌بست در تحلیل ایمنی سیستم‌های نرم‌افزاری

عین‌الله پیرا

چکیده: تحلیل ایمنی سیستم‌های نرم‌افزاری، خصوصاً از نوع بحرانی- ایمنی، باید به طور دقیق انجام شود زیرا وجود حتی یک خطای کوچک در چنین سیستم‌هایی ممکن است نتایج فاجعه‌باری داشته باشد. ضمناً چنین تحلیلی باید قبل از پیاده‌سازی یعنی در مرحله طراحی و در سطح مدل انجام شود. وارسی مدل، یک روش دقیق و مبتنی بر ریاضی است که ایمنی سیستم‌های نرم‌افزاری را با دریافت مدلی از آن و بررسی تمام حالت‌های قابل دسترس مدل انجام می‌دهد. با توجه به پیچیدگی بعضی سیستم‌ها و مدل‌های آن، وارسی مدل ممکن است با مشکل انفجار فضای حالت مواجه شود، یعنی نتواند تمام حالت‌های قابل دسترس را پیمایش کند. یک راه حل برای حل مشکل انفجار فضای حالت در چنین سیستم‌هایی این است که به جای تأیید ایمنی، وارسی مدل سعی می‌کند آنها را با یافتن خطاهایی از جمله بن‌بست (در صورت موجود) رد کند. اگرچه قبلاً هیوریستیکی برای یافتن بن‌بست در فضای حالت مدل ارائه شده و آن را در چندین الگوریتم جستجوی مکاشفه‌ای ساده و تکاملی به کار برده‌اند ولی سرعت تشخیص آن پایین بوده است. این مقاله، یک هیوریستیک جدید برای یافتن بن‌بست در فضای حالت مدل ارائه می‌کند و سرعت تشخیص آن با به کار بردن در الگوریتم‌های جستجوی مکاشفه‌ای ساده از جمله عمقی تکرارشونده *A و جستجوی پرتو و الگوریتم‌های تکاملی مختلف از جمله ژنتیک، بهینه‌سازی ازدحام ذرات و بهینه‌سازی بیزی با روش قبلی مقایسه می‌شود. مقایسه نتایج تجربی تأیید می‌کنند که هیوریستیک جدید می‌تواند حالت بن‌بست را در زمان کمتری نسبت به هیوریستیک قبلی پیدا کند.

کلیدواژه: **الگوریتم‌های تکاملی، بن‌بست، تحلیل ایمنی، وارسی مدل، هیوریستیک.**

۱- مقدمه

سیستم‌های نرم‌افزاری مدرن شامل زیرسیستم‌هایی می‌باشند که در بعضی شرایط مجبور هستند به منابع سیستم به طور هم‌روند و مشترک دسترسی پیدا کنند و این ممکن است باعث ایجاد حالت بن‌بست[1] شود. کشف و رفع حالت‌های خطا (برای مثال بن‌بست) یا به عبارت کلی‌تر، تحلیل ایمنی سیستم‌های نرم‌افزاری، خصوصاً از نوع بحرانی- ایمنی[2]، باید به طور دقیق و قبل از پیاده‌سازی یعنی در مرحله طراحی و در سطح مدل انجام شود. مجموعه‌ای از روش‌های رسمی[3] مبتنی بر تحلیل‌های دقیق ریاضی وجود دارند که صحت مدل را با توجه به ویژگی‌های مورد انتظار بررسی می‌کنند. وارسی مدل[4] یک روش رسمی است که مدلی از سیستم به همراه ویژگی‌های توصیف‌شده در قالب فرمول‌های منطق زمانی[5] را دریافت کرده و درستی/ نادرستی ویژگی‌ها را با کاوش همه حالت‌های قابل دسترسِ (فضای حالت) مدل بررسی می‌کند [۱]. با توجه به پیچیدگی بعضی سیستم‌ها و مدل‌های آن، وارسی مدل ممکن است با مشکل انفجار فضای حالت (کمبود حافظه در تولید همه حالت‌های ممکن) مواجه شود. یک راه حل برای حل مشکل انفجار فضای حالت در چنین سیستم‌هایی این است که وارسی مدل سعی می‌کند آنها را با یافتن خطاهایی از جمله بن‌بست (در صورت موجود) رد کند. به این مفهوم که به جای اثبات عدم وجود خطا، دنبال خطا می‌گردد. لازم به ذکر است که اگر وارسی مدل در چنین حالتی نتواند خطایی پیدا را کند، نمی‌توان ادعا کرد که سیستم بدون خطا است زیرا فقط قسمتی از فضای حالت پیمایش شده است [۲]. البته می‌توان با روش‌های احتمالی از جمله مونت‌کارلو تخمین معناداری را از میزان صحت سیستم ارائه کرد [۳].

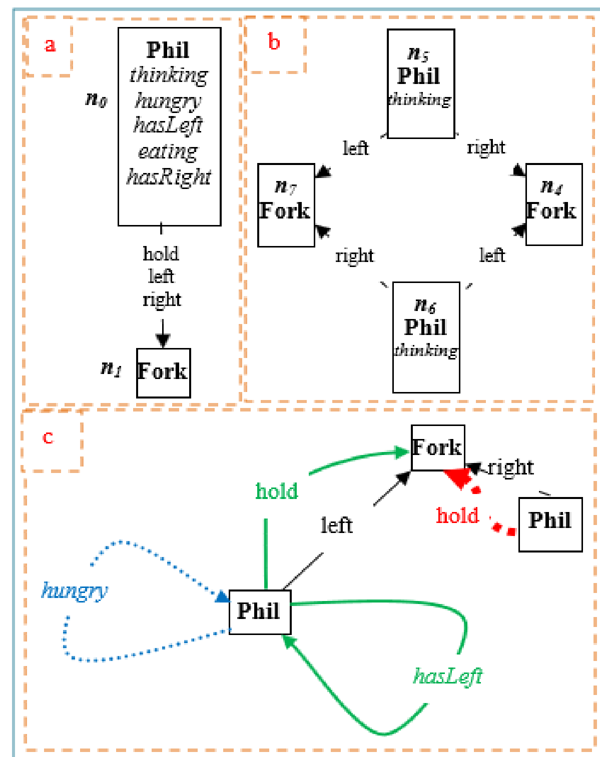
فضای حالت مدل معمولاً با یک گراف نمایش داده می‌شود که در آن، رئوس و یال‌ها به ترتیب مجموعه حالت‌ها و گذارهای مابین آنها را نشان می‌دهند. در فضای حالت مدل، بن‌بست، حالتی را نشان می‌دهد که هیچ گذار خروجی ندارد که به چنین حالتی، حالت نهایی نیز می‌گویند. به عبارت دیگر، چنین گرهی در گراف معادل، یال خروجی ندارد. بعد از عمل وارسی، وارسی‌گر یک مثال نقض[6] تولید می‌کند. یک مثال نقض نشان‌دهنده مسیری از حالت‌ها (گره‌ها) و گذارهای (یال‌های) مابین آنها است به طوری که این مسیر از حالت (گره) ابتدایی شروع شده و به یک حالت (گره) بن‌بست ختم می‌شود. یافتن چنین مسیری نیز ممکن است باعث بروز مشکل انفجار فضای حالت شود. در [۴] هیوریستیکی برای یافتن بن‌بست در فضای حالت مدل ارائه شده است. این هیوریستیک مسیری را دریافت کرده و مجموع تعداد گذارهای خروجی گره‌های موجود در مسیر مورد نظر را برمی‌گرداند. واضح است که هرچه مقدار هیوریستیک برای مسیری کمتر باشد، احتمال به بن‌بست رسیدن آن بالا خواهد بود. برای تست این هیوریستیک، آن را در الگوریتم‌های جستجوی عمقی[7] (DFS) و ردیفی[8] (BFS) [۵]، جستجوی مکاشفه‌ای ساده از جمله *A [۵]، عمقی تکرارشونده *A (*IDA) و جستجوی پرتو[9] (BS)

این مقاله در تاریخ ۲۰ اردیبهشت ماه ۱۴۰۰ دریافت و در تاریخ ۲ آذر ماه ۱۴۰۰ بازنگری شد.

عین‌الله پیرا (نویسنده مسئول)، دانشکده فناوری اطلاعات و مهندسی کامپیوتر، دانشگاه شهید مدنی آذربایجان، تبریز، ایران، (email: pira@azaruniv.ac.ir).

1. Deadlock
2. Safety-Critical
3. Formal Methods
4. Model Checking
5. Temporal Logic
6. Counterexample
7. Depth First Search
8. Breadth First Search
9. Beam Search

www.SID.ir



شکل ۱: جزئیات مدل طراحی‌شده برای مسأله غذاخوری فیلسوفان با ۲ فیلسوف [۶].

[۶] و الگوریتم‌های تکاملی مختلف از جمله ژنتیک (GA)[1] [۴]، [۷] و [۸]، بهینه‌سازی کلونی مورچگان (ACO)[2] [۹]، شبیه‌سازی ذوب فلزات (SA)[3] [۱۰]، بهینه‌سازی ازدحام ذرات (PSO)[4] [۱۱] و [۱۲] و بهینه‌سازی بهینه‌سازی بیزی (BOA)[5] [۶] به کار برده‌اند. اگرچه این الگوریتم‌ها می‌توانند حالت بن‌بست را با موفقیت تشخیص دهند ولی سرعت تشخیص می‌تواند بهبود پیدا کند.

در این مقاله، یک هیوریستیک جدید برای یافتن بن‌بست در فضای حالت مدل ارائه می‌کنیم. این هیوریستیک به هر مسیری که تعداد گذارهای خروجی گره‌های موجود در آن تقریباً نزولی باشد، عدد کمتری را اختصاص می‌دهد. در واقع به جای مجموع تعداد گذارهای خروجی گره‌های موجود، مجموع حاصل‌ضرب اختلاف تعداد گذارهای گره‌های پشت سر هم (s_{i-1} و s_i) در موقعیت گره s_i یعنی i را به عنوان مقدار برازندگی برای آن مسیر در نظر می‌گیرد. هرچه این مقدار برای مسیری کمتر باشد (یعنی تعداد گذارهای خروجی گره‌های موجود در آن تقریباً نزولی باشد)، احتمال رسیدن آن مسیر به بن‌بست بیشتر خواهد بود. برای تست هیوریستیک جدید، آن را در الگوریتم‌های مختلف استفاده کرده و این الگوریتم‌ها را در ابزار GROOVE [۱۳] پیاده‌سازی می‌کنیم. سپس با اجرای این الگوریتم‌ها روی چندین مسأله شناخته‌شده، سرعت تشخیص آن را با هیوریستیک قبلی مقایسه می‌کنیم. قابل ذکر است که ابزار GROOVE، مدل‌های توصیف‌شده با زبان رسمی تبدیل گراف [۱۴] را می‌پذیرد و در نتیجه مسایل در نظر گرفته شده نیز باید با این زبان رسمی توصیف شوند.

1. Genetic Algorithm
2. Ant Colony Optimization
3. Simulated Annealing
4. Particle Swarm Optimization
5. Bayesian Optimization Algorithm

Archive of SID

۲- پیش‌زمینه

۲-۱ زبان رسمی تبدیل گراف

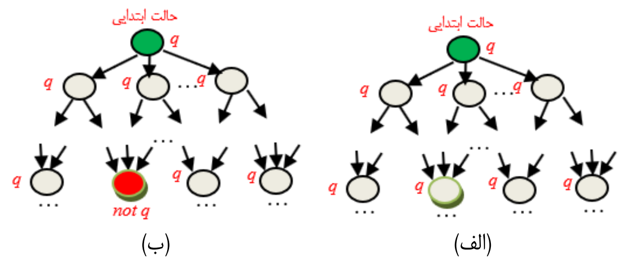
این زبان از گراف‌ها و تبدیلات مابین آنها جهت مدل‌سازی حالت‌ها و رفتارهای یک سیستم بهره می‌برد [۱۴]. مدل‌هایی که با این زبان توصیف می‌شوند دارای سه جزء گراف نوع TG، گراف میزبان HG و مجموعه قوانین تبدیل R است. گراف TG شامل مجموعه‌ای از نوع‌های گره[6] و مجموعه‌ای از نوع‌های یال[7] است و گره‌ها و یال‌های مجاز را نشان می‌دهد که می‌توانند در طی تبدیلات گراف وجود داشته باشند. گراف HG حالت اولیه (آغازین) سیستم را نشان می‌دهد و بدیهی است که گره‌ها و یال‌های آن باید جزء نوع‌های گره و نوع‌های یال موجود در گراف TG باشند. مجموعه R قوانین تبدیلی را نشان می‌دهد که حالات مختلف سیستم را می‌توانند تولید کنند. هر قانون تبدیل r به صورت سه‌تایی (LHS, RHS, NAC) تعریف می‌شود که در آن، گراف LHS شرایطی را مشخص می‌کند که باید گراف مربوط به حالت فعلی داشته باشد تا این قانون بتواند روی این حالت اِعمال شود. گراف RHS نیز شرایطی را تعیین می‌کند که باید گراف مربوط به حالت حاصل بعد از اِعمال قانون داشته باشد. گراف NAC نیز محدودیت‌هایی را مشخص می‌کند که نباید در گراف مربوط به حالت فعلی باشند تا این قانون بتواند اِعمال شود. شکل ۱ جزئیات مدل طراحی‌شده برای مسأله غذاخوری فیلسوفان با ۲ فیلسوف را نشان می‌دهد [۶]. قسمت‌های a و b در این شکل به ترتیب گراف‌های نوع و میزبان را نشان می‌دهند. مطابق این گراف نوع، هر فیلسوف می‌تواند در حالت‌های مختلف فکرکردن (thinking)، گرسنه‌بودن (hungry)، فقط چنگال چپ را در دست داشتن (hasLeft)، هر دو چنگال را در دست داشتن و خوردن (eating) و فقط چنگال راست در دست داشتن (hasRight) باشد. گراف میزبان نیز نشان می‌دهد فیلسوفان در ابتدا در حالت فکرکردن هستند. قسمت c قانون get-left را نشان می‌دهد. در گراف مربوط به این قانون، گراف‌های LHS، RHS و NAC را با هم ادغام کرده و از کدگذاری رنگی برای تشخیص هر کدام از اجزای آن استفاده می‌کنیم. اجزای مشترک بین گراف‌های LHS و RHS به صورت نازک سیاه‌رنگ هستند. اجزای مختص گراف LHS به صورت گره‌ها با دو لبه و یال‌های نقطه‌چین آبی‌رنگ مشخص گردیده‌اند در حالی که اجزای مختص گراف RHS با رنگ سبز مشخص شده‌اند. در نهایت گره‌ها با دو لبه و یال‌های نقطه‌چین قرمزرنگ متعلق به گراف NAC هستند. برای اِعمال قانون $r = (LHS, RHS, NAC)$ روی حالت s (که در واقع به صورت یک گراف است)، ابتدا رخدادهای گراف LHS در حالت s پیدا شده و یکی از آنها که شامل گراف NAC نیست با گراف RHS جایگزین شده و حالت جدیدی را تولید می‌کند. برای تولید فضای حالت یک مدل، همه قوانین باید به طور مکرر روی حالت‌های تولیدشده اِعمال شوند.

۲-۲ وارسی مدل

وارسی مدل یک روش رسمی مبتنی بر تحلیل دقیق ریاضی است که درستی/ نادرستی مدل را با توجه به ویژگی‌های توصیف‌شده در قالب فرمول‌های منطق زمانی بررسی می‌کند. ایمنی یکی از ویژگی‌های مهم سیستم است که می‌تواند از طریق وارسی مدل و با بررسی تمام حالت‌های

6. Node Types
7. Edge Types

www.SID.ir



شکل ۲: (الف) تأیید و (ب) انکار ویژگی ایمنی q.

قابل دسترس سیستم تأیید شود. شکل ۲- الف مثالی از تأیید ایمنی ویژگی q را در فضای حالت یک مدل نشان می‌دهد که مطابق این شکل، ویژگی q باید در تمام حالت‌ها ارضا شود. با توجه به این که تعداد حالت‌های قابل دسترس در سیستم‌های بزرگ باعث وقوع مشکل انفجار فضای حالت می‌شوند، بنابراین وارسی مدل سعی می‌کند به جای تأیید ویژگی ایمنی q، آن را انکار کند [۱]. انکار ویژگی ایمنی q می‌تواند با یافتن حالتی در فضای حالت و نقض ویژگی q محقق شود. به عبارت دیگر، حالتی در فضای حالت یافت بشود که در آن ویژگی $\text{not}\, q$ ارضا گردد. شکل ۲- ب این روند را توضیح می‌دهد. به محض پیداشدن چنین حالتی (حالت هدف)، وارسی‌گر یک مثال نقض تولید می‌کند که شامل اطلاعاتی از جمله محل واقعی خطا در سیستم است و کاربر می‌تواند با استفاده از این اطلاعات، خطای گزارش‌شده را در سیستم رفع کند. ولی اگر چنین حالتی (حالت هدف) پیدا نشود نمی‌توان ادعا کرد که سیستم بدون خطا است زیرا فقط قسمتی از فضای حالت پیمایش شده است.

۲-۳ خلاصه‌ای از الگوریتم‌های به کار رفته در این مقاله

در این بخش، خلاصه‌ای از الگوریتم‌های به کار رفته در این مقاله از جمله عمقی تکرارشونده A* (IDA*) و جستجوی پرتو (BS) و الگوریتم‌های تکاملی مختلف از جمله ژنتیک (GA)، بهینه‌سازی ازدحام ذرات (PSO) و بهینه‌سازی بیزی (BOA) را ارائه می‌کنیم.

الگوریتم عمقی تکرارشونده A*: این الگوریتم ترکیبی از الگوریتم‌های جستجوی عمق اول (DFS) و A* [۱۵] را به شکل مداوم تکرار کرده تا یا حالت هدف پیدا شود و یا تعداد تکرارها (عمق جستجو) به عدد از پیش تعیین شده $maxDepth$ برسد. در تکرار iاُم ($1 \le i \le maxDepth$)، جستجوی عمقی تا عمق i را انجام داده و هزینه این مسیرها را مشابه الگوریتم A* و با استفاده از هیوریستیک ارائه‌شده محاسبه می‌کند و مسیرهایی را که هزینه آنها از یک مقدار آستانه بیشتر باشد نادیده می‌گیرد.

جستجوی پرتو: این الگوریتم مشابه جستجوی سطح اول (BFS) به صورت سطح به سطح فضای حالت را کاوش می‌کند با این تفاوت که فقط به تعداد مشخصی، حالت (عرض پرتو) با بیشترین مقدار هیوریستیک را در هر سطح در نظر گرفته و بقیه حالت‌ها را در نظر نمی‌گیرد. این الگوریتم تا موقعی ادامه پیدا می‌کند که یا حالت هدف پیدا شود و یا تعداد سطوح پیمایش‌شده (عمق جستجو) به یک عدد از پیش مشخص شده $maxDepth$ برسد.

الگوریتم ژنتیک: این الگوریتم از مشهورترین الگوریتم‌های تکاملی می‌باشد که بر اصل بقای اصلح داروین استوار است [۱۶] و همواره با جمعیتی از راه حل‌های کاندیدا (اصطلاحاً کروموزوم‌ها) سروکار دارد که در آن جمعیت اولیه به صورت تصادفی تولید می‌شود. این الگوریتم برای تولید جمعیت بعدی، ۱) جمعیت فعلی را با استفاده از یک تابع برازش ارزیابی کرده و تعدادی از برازنده‌ترین آنها را انتخاب می‌کند. ۲) به کمک عملگر ترکیب و با احتمال نرخ ترکیب، کروموزوم‌های انتخاب‌شده را با هم ترکیب کرده تا کروموزوم‌های جدید تولید شوند. ۳) با احتمال نرخ جهش، بعضی کروموزوم‌های جدیداً تولیدشده را انتخاب کرده و بعضی ژن‌های آنها را تغییر می‌دهد. ۴) کروموزوم‌های جدیداً تولیدشده را با بعضی از کروموزوم‌های فعلی که دارای شایستگی پایینی می‌باشند جایگزین می‌کند. این روند (تولید جمعیت بعدی از فعلی) تا هنگامی ادامه پیدا می‌کند که یک راه حل (تقریباً) بهینه پیدا شود و یا تعداد تکرارها به یک عدد بیشینه برسد. در مسأله وارسی ایمنی و پیداکردن یک حالت بن‌بست توسط این الگوریتم، هر مسیری که از حالت ابتدایی شروع شده و دارای طول مشخصی ($maxDepth$) باشد به عنوان یک کروموزوم در نظر گرفته می‌شود. همچنین هیوریستیک برای یافتن بن‌بست به عنوان تابع برازش در نظر گرفته می‌شود [۷].

الگوریتم بهینه‌سازی ازدحام ذرات: این الگوریتم مبتنی بر رفتار اجتماعی پرندگان است که به صورت گروهی دنبال غذا می‌گردند [۱۷] و با گروهی از ذرات (راه حل‌ها) سروکار دارد که در ابتدا به صورت تصادفی تولید می‌شوند. هر ذره دارای یک موقعیت و سرعت حرکت در فضای جستجو است. موقعیت، سرعت و جهت حرکت یک ذره تحت تأثیر بهترین موقعیت خود ($pbest$) و موقعیت دیگران ($gbest$) قرار دارد. بعد از تولید تصادفی ذرات در مرحله ابتدایی، میزان برازندگی آنها با استفاده از تابع برازش محاسبه شده و مقادیر $pbest$ و $gbest$ به روز رسانی می‌شوند. در هر مرحله تکرار، موقعیت و سرعت ذرات توسط (۱) و (۲) به روز رسانی گردیده و این روند تا موقعی ادامه می‌یابد که یک راه حل (تقریباً) بهینه پیدا شود و یا تعداد تکرارها به یک عدد بیشینه برسد. مشابه با الگوریتم ژنتیک، در این الگوریتم نیز هر مسیری که از حالت ابتدایی شروع شده و دارای طول مشخصی ($maxDepth$) باشد به عنوان یک ذره در نظر گرفته می‌شود [۱۱]

$$v[t+1] = w \times v[t] + C1 \times rnd \times (pbest[t] - position[t]) + C2 \times rnd \times (gbest[t] - position[t]) \quad (1)$$

$$position[t+1] = position[t] + v[t+1] \quad (2)$$

مقدار w تأثیر سرعت فعلی ذره را بر سرعت بعدی آن مشخص می‌کند. مقادیر $C1$ و $C2$ به ترتیب تأثیر بهترین موقعیت خود و دیگران را بر سرعت بعدی ذره تعیین می‌کنند و مقادیر آنها طوری در نظر گرفته می‌شود که مجموع آنها حداکثر برابر ۴ شود.

الگوریتم بهینه‌سازی بیزی: این الگوریتم نوعی الگوریتم تخمین توزیع است که در آن، یک شبکه بیزی از راه حل‌های برازنده جمعیت فعلی ساخته شده و همچنین این شبکه نیز برای تولید راه حل‌های جدید نمونه‌برداری می‌شود [۱۸]. شبکه بیزی یک مدل گرافیکی احتمالی است که وابستگی‌های شرطی مابین متغیرهای مختلف مسأله را نمایش می‌دهد [۱۹]. یک شبکه بیزی دارای دو مؤلفه ساختار و پارامترها می‌باشد: ساختار توسط یک گراف جهت‌دار بدون دور نمایش داده می‌شود که در آن، رئوس و یال‌ها به ترتیب متغیرهای مسأله و وابستگی‌های شرطی مابین آنها را نشان می‌دهند. پارامترهای شبکه نیز توسط مجموعه‌ای از جداول احتمال شرطی مشخص می‌شوند به این صورت که هر جدول، احتمال‌های شرطی همه مقادیر یک متغیر وابسته را به ازای همه مقادیر والدهای (رئوس آغازین یال‌ها) آن متغیر نشان می‌دهد. الگوریتم بهینه‌سازی بیزی پس از تولید تصادفی جمعیت اولیه، آنها را با استفاده از تابع برازش ارزیابی کرده و با راه حل‌های (کروموزوم‌های) برازنده انتخاب‌شده، یک شبکه بیزی ایجاد می‌کند. سپس از روش نمونه‌برداری منطق احتمالی استفاده کرده و کروموزوم‌های جدید تولید می‌نماید و آنها را با استفاده از یک

Archive of SID
www.SID.ir

Archive of SID

روش جایگزینی به جمعیت فعلی وارد می‌کند. این فرایند تا موقعی تکرار می‌گردد که یک راه حل (تقریباً) بهینه پیدا شود و یا تعداد تکرارها به یک عدد بیشینه برسد. مشابه با الگوریتم ژنتیک، در این الگوریتم نیز هر مسیری که از حالت ابتدایی شروع شده و دارای طول مشخصی (*maxDepth*) باشد به عنوان یک کروموزوم در نظر گرفته می‌شود [۶].

۳- هیوریستیک ارائه‌شده

همان طور که اشاره شد بن‌بست در فضای حالت یک مدل به حالتی گفته می‌شود که هیچ گذار خروجی ندارد. در مدل‌های طراحی‌شده با زبان تبدیل گراف می‌توان حالت بن‌بست را به صورت زیر بیان کرد: حالت s بن‌بست است اگر برای همه قوانین به شکل $r=(LHS,RHS,NAC)$، حالت s دارای هیچ رخدادی از گراف LHS نباشد یا رخداد موجود شامل گراف NAC باشد. در مسیری که از حالت ابتدایی شروع شده و به حالت بن‌بست ختم می‌شود، ابتدا تعداد زیادی قانون می‌توانند روی حالت ابتدایی اِعمال شوند ولی روی حالت‌های بعدی (خصوصاً حالت‌های انتهایی مسیر)، تعداد قوانین اِعمال‌شده (گذارهای خروجی) کمتر گردیده و در نهایت به صفر می‌رسند. هیوریتسیکی که قبلاً برای یافتن چنین مسیری ارائه شده است (به نام HeuSum) فقط مجموع تعداد گذارهای خروجی گره‌های موجود در مسیر مورد نظر را به عنوان برازندگی در نظر می‌گیرد. به عنوان مثال برای دو مسیر با تعداد گذارهای خروجی $path۱:۳,۲,۱,۲,۲$ و $path۲:۳,۲,۳,۲,۱$ به ترتیب مقادیر ۱۰ و ۱۱ را در نظر می‌گیرد و مفهومش این است که مسیر ۱ شانس بیشتری برای رسیدن به حالت بن‌بست دارد. در حالی که احتمال رسیدن مسیر ۲ به بن‌بست از مسیر ۱ بیشتر است زیرا تعداد گذارهای خروجی برای حالت‌های موجود رفته‌رفته به عدد صفر نزدیک می‌شود.

هیوریستیک ارائه‌شده در این مقاله (به نام HeuSumDiff) به جای مجموع تعداد گذارهای خروجی، مجموع حاصل‌ضرب اختلاف تعداد گذارهای گره‌های پشت سر هم (s_{i-1} و s_i) در موقعیت گره s_i یعنی i را به عنوان مقدار برازندگی برای آن مسیر در نظر می‌گیرد. هرچه این مقدار برای مسیری کمتر باشد (یعنی تعداد گذارهای خروجی گره‌های موجود در آن تقریباً نزولی باشد)، احتمال رسیدن آن مسیر به بن‌بست بیشتر خواهد بود. فرض کنید که می‌خواهیم مقدار برازندگی را برای مسیر $path="s_1s_2\ldots s_L"$ با استفاده از هیوریستیک ارائه‌شده محاسبه کنیم. اگر n_i تعداد گذارهای خروجی برای حالت s_i باشد، مقدار برازندگی برای مسیر $path$ از فرمول زیر محاسبه می‌شود

$$fit(path)=\sum_{i=۱}^{L} i\times(n_i-n_{i-۱}) \tag{۳}$$

در مسیرهای منجر به بن‌بست، چون مقادیر $n_i-n_{i-۱}$ برای حالت‌های آخر (iهای نزدیک به L) اعداد منفی نزدیک صفر هستند، جهت افزایش تأثیر این مقادیر در مقدار برازندگی، i برابر این مقادیر را در نظر می‌گیریم. به عنوان مثال، این تابع برای دو مسیر با تعداد گذارهای خروجی $path۱:۳,۲,۱,۲,۲$ و $path۲:۳,۲,۳,۲,۱$ به ترتیب مقادیر ۰ و ۶- را به صورت زیر محاسبه می‌کند و این نشان می‌دهد که مسیر ۲ برازنده‌تر است

$$\begin{aligned}&fit(path۱)=\\&۱\times(۲-۳)+۲\times(۱-۲)+۳\times(۲-۱)+۴\times(۲-۲)=۰\\&fit(path۲)=\\&۱\times(۲-۳)+۲\times(۳-۲)+۳\times(۲-۳)+۴\times(۱-۲)=-۶\end{aligned} \tag{۴}$$

۴- ارزیابی و مقایسه کارایی هیوریستیک‌های HEUSUMDIFF و HEUSUM

برای ارزیابی و مقایسه کارایی هیوریستیک‌های HeuSumDiff و HeuSum، آنها را در الگوریتم‌های *IDA، BS، GA، PSO و BOA استفاده کرده و این الگوریتم‌ها را با زبان برنامه‌نویسی Java و در ابزار متن‌باز GROOVE پیاده‌سازی می‌کنیم. سپس با اجرای این الگوریتم‌ها روی مسایل غذاخوری فیلسوفان [۲۰]، پروتکل انتقال مجدد به تعداد محدود [۲۱] و چرخه حیات پردازه [۲۲]، سرعت تشخیص آنها را با هم مقایسه می‌نماییم. نسخه جدید ابزار GROOVE به همراه مسایل استفاده‌شده روی وب قابل دسترس است[۱].

مسأله غذاخوری فیلسوفان (DPH): این مسأله، نحوه غذاخوری چندین فیلسوف دور یک میز را توصیف می‌کند که مابین هر دوی آنها یک چنگال وجود دارد. هر فیلسوف می‌تواند در حالت‌های مختلف فکرکردن (thinking)، گرسنه‌بودن (hungry)، فقط چنگال چپ را در دست داشتن (hasLeft)، هر دو چنگال را در دست داشتن و خوردن (eating) و فقط چنگال راست در دست داشتن (hasRight) باشد. در ابتدا همه فیلسوفان در حالت thinking هستند و می‌توانند بعد از مدتی احساس گرسنگی کرده و به حالت hungry بروند. فیلسوف با حالت hungry می‌تواند چنگال چپ را در صورت آزادبودن برداشته و به حالت hasLeft برود و اگر چنگال راستش نیز آزاد باشد، آن را برداشته و به حالت eating می‌رود. فیلسوف در حال eating بعد از مدتی چنگال چپش را رها کرده و به حالت hasRight می‌رود و سپس چنگال راستش را نیز رها کرده و دوباره به حالت thinking می‌رود. در این مسأله اگر همه فیلسوفان به طور هم‌زمان چنگال چپ را برداشته و به حالت hasLeft بروند و منتظر چنگال راستشان باشند، آن گاه حالت بن‌بست پیش می‌آید.

مسأله پروتکل انتقال مجدد به تعداد محدود (BRP): این پروتکل که مربوط به انتقال فایل‌ها از طریق یک کانال ارتباطی lossy است توسط شرکت فیلیپس استفاده می‌گردد. در این پروتکل، برخی فریم‌های فایل انتقالی می‌توانند به دفعات محدودی مجدداً انتقال شوند. اگر بعضی فریم‌ها با مشکل مواجه گردند ممکن است انتقال فایل لغو شود و چنین وضعیتی را می‌توان به عنوان یک حالت بن‌بست (نهایی) تلقی کرد.

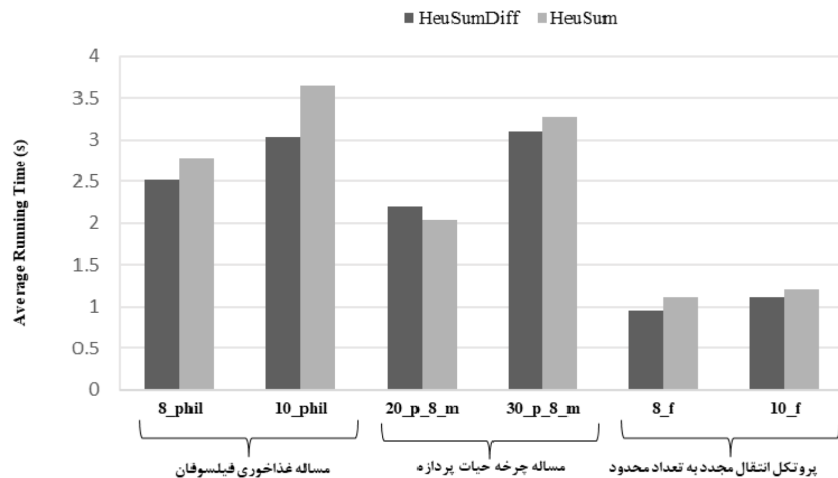
مسأله چرخه حیات پردازه (PLC): در این مسأله، چرخه حیات یک فرایند در یک سیستم عامل از مرحله تولد (ایجاد) تا مرگ (خاتمه) توصیف می‌شود. فرایند ایجادشده در صورت وجود فضای کافی، به حافظه لود شده و منتظر ابزار CPU یا I/O می‌ماند. به محض اتمام اجرای فرایند، تمام منابع در دسترس آزاد شده و فرایند خاتمه می‌یابد. در این مسأله، اجرا و خاتمه همه فرایندهای موجود را می‌توان به عنوان وضعیت بن‌بست (نهایی) در نظر گرفت.

قبل از اجرای الگوریتم‌ها باید مقادیر مناسب را برای پارامترهای مورد نیاز آنها ارائه کنیم. بعضی پارامترها برای مدل‌ها با اندازه‌های مختلف دارای مقدار ثابتی هستند که جدول ۱ مقادیر مناسب را برای این پارامترهای ثابت نشان می‌دهد. در حالی که بعضی پارامترها از جمله اندازه جمعیت (population) و حداکثر عمق جستجو یا همان طول کروموزوم‌ها (*maxDepth*) بسته به نوع مسأله و اندازه مدل دارای مقادیر متغیری می‌باشند. جدول ۲ مقادیر مناسب را برای این پارامترهای متغیر نشان می‌دهد.

1. https://github.com/EinPira/Groove-HEUSUMDIFF

www.SID.ir

Archive of SID



شکل ۳: مقایسه میانگین زمان اجرای الگوریتم بهینه‌سازی بیزی برای کشف بن‌بست.

جدول ۱: مقادیر مناسب برای پارامترهای ثابت الگوریتم‌ها.

حداکثر تعداد تکرار هر الگوریتم		۱۰۰	
الگوریتم بهینه‌سازی بیزی (BOA)		جستجوی پرتو (BS)	
نرخ انتخاب (selecton rate)	۰/۴	مسایل غذاخوری فیلسوفان و چرخه حیات پردازه	عرض پرتو = ۱۰
نرخ جایگزینی (replacement rate)	۰/۵	پروتکل انتقال مجدد به تعداد محدود	عرض پرتو = ۲۰
الگوریتم ژنتیک (GA)		الگوریتم بهینه‌سازی ازدحام ذرات (PSO)	
نرخ ترکیب (crossover rate)	۰/۶	$C1, C2$	۲
نرخ جهش (mutation rate)	۰/۳	w	۰/۸

جدول ۲: مقادیر مناسب برای پارامترهای متغیر الگوریتم‌ها.

مسأله	اندازه مدل	اندازه جمعیت	حداکثر عمق جستجو
غذاخوری فیلسوفان	۸ فیلسوف	۱۰	۲۰
	۱۰ فیلسوف	۱۵	۲۵
چرخه حیات پردازه	۲۰ فرایند و ۸ قطعه حافظه	۲۰	۱۸۰
	۳۰ فرایند و ۸ قطعه حافظه	۴۰	۲۸۰
پروتکل انتقال مجدد به تعداد محدود	۸ فریم	۲۰	۵۰
	۱۰ فریم	۳۰	۶۰

الگوریتم‌های استفاده‌شده در این مقاله، قبلاً در مقالات مختلفی ارائه شده‌اند و دقت آنها در تشخیص حالت بن‌بست، روی مسایل مختلف و با اندازه‌های متفاوت بررسی گردیده‌اند. مطابق با مقالات مرتبط با این الگوریتم‌ها، میزان دقت (accuracy) هر کدام از الگوریتم‌ها به پارامترهای مختلفی از جمله حداکثر عمق جستجو وابسته است. با اجراهای مختلف این الگوریتم‌ها به ازای هیوریستیک‌های HeuSumDiff و HeuSum به این نتیجه رسیده‌ایم که این هیوریستیک‌ها دارای دقت مشابهی هستند، یعنی اگر الگوریتمی با هیوریستیک HeuSum بتواند بن‌بستی را پیدا کند با هیوریستیک HeuSumDiff نیز خواهد توانست و بالعکس. در این مقاله، مسایل (غذاخوری فیلسوفان، پروتکل انتقال مجدد به تعداد محدود و چرخه حیات پردازه) و مدل‌هایی را انتخاب کرده‌ایم که این الگوریتم‌ها بتوانند به طور ۱۰۰٪ حالت بن‌بست را پیدا کرده تا امکان مقایسه سرعت تشخیص این هیوریستیک‌ها فراهم شود.

برای تولید نتایج، کلیه الگوریتم‌ها را با استفاده از یک پردازنده Core i۵ Duo p۸۴۰۰(۲/۵GHz) ۶GB RAM اجرا کرده‌ایم. شکل‌های ۳ تا ۷ نمودار میانگین حاصل از اجرای این الگوریتم‌ها به تعداد ۳۰ بار را برای کشف حالت بن‌بست در مدل‌ها با اندازه‌های مختلفی از مسایل غذاخوری فیلسوفان، پروتکل انتقال مجدد به تعداد محدود و چرخه حیات پردازه نمایش می‌دهد. در این نمودارها، برای مدل‌ها با اندازه x فیلسوف از نماد x_phil، برای مدل‌ها با اندازه x فرایند و y قطعه حافظه از نماد $x_p_y_m$ و برای مدل‌ها با اندازه x فریم از نماد x_f استفاده می‌کنیم.

مطابق شکل ۳، در اکثر مدل‌های در نظر گرفته شده، میانگین زمان اجرای الگوریتم BOA برای هیوریستیک HeuSumDiff کمتر از زمان اجرای آن به ازای هیوریستیک HeuSum است. در مدل $۲۰_p_۸_m$ از مسأله چرخه حیات پردازه، هیوریستیک HeuSum دارای سرعت تشخیص بهتری نسبت به هیوریستیک HeuSumDiff است.

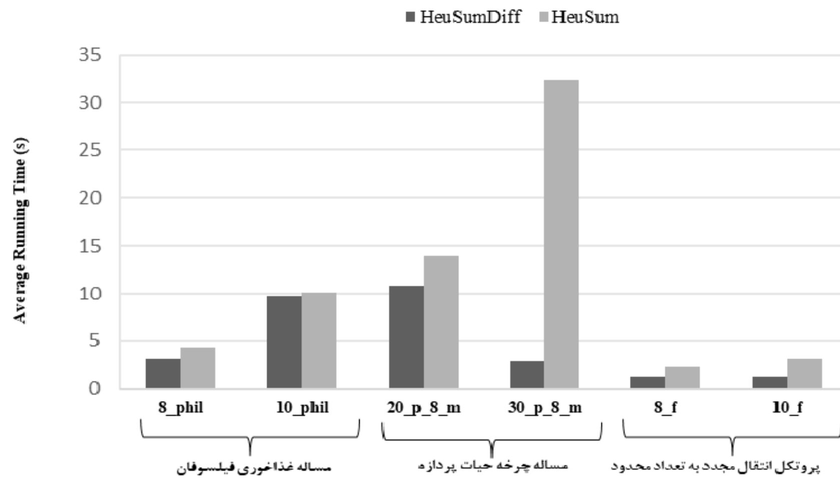
برای الگوریتم GA (شکل ۴)، سرعت تشخیص بن‌بست در همه مدل‌های در نظر گرفته شده به ازای هیوریستیک HeuSumDiff بیشتر از هیوریستیک HeuSum است. در همه مدل‌ها به جز مدل $۳۰_p_۸_m$ از مسأله چرخه حیات پردازه، تفاوت سرعت تشخیص این دو هیوریستیک نسبتاً کم است ولی در این مدل، هیوریستیک HeuSumDiff سرعت تشخیص خیلی بهتری نسبت به هیوریستیک HeuSum دارد.

مشابه با الگوریتم‌های BOA و GA، سرعت تشخیص بن‌بست توسط الگوریتم PSO در همه مدل‌های در نظر گرفته شده، به ازای هیوریستیک HeuSumDiff بیشتر از هیوریستیک HeuSum است (شکل ۵). البته در همه مدل‌های مسأله چرخه حیات پردازه، تفاوت سرعت تشخیص این دو هیوریستیک خیلی زیاد است.

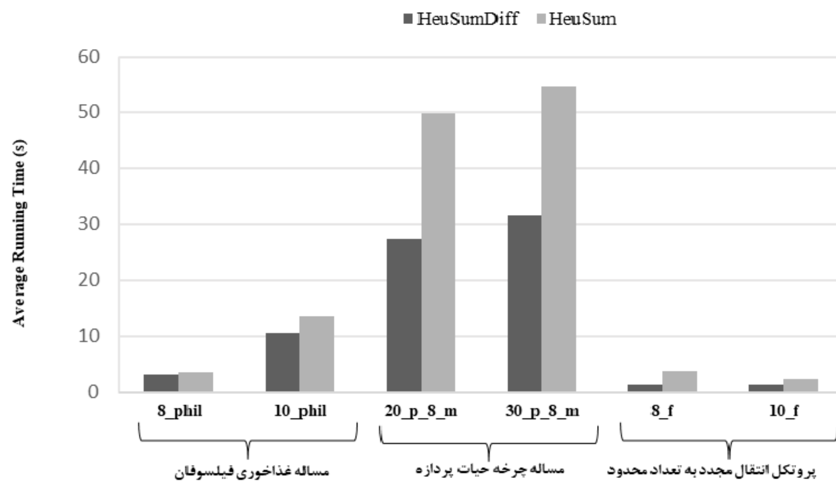
شکل‌های ۶ و ۷ تأیید می‌کنند که سرعت تشخیص بن‌بست توسط الگوریتم‌های *IDA و BS در همه مدل‌های در نظر گرفته شده به ازای هیوریستیک HeuSumDiff کمی بیشتر از هیوریستیک HeuSum است.

www.SID.ir

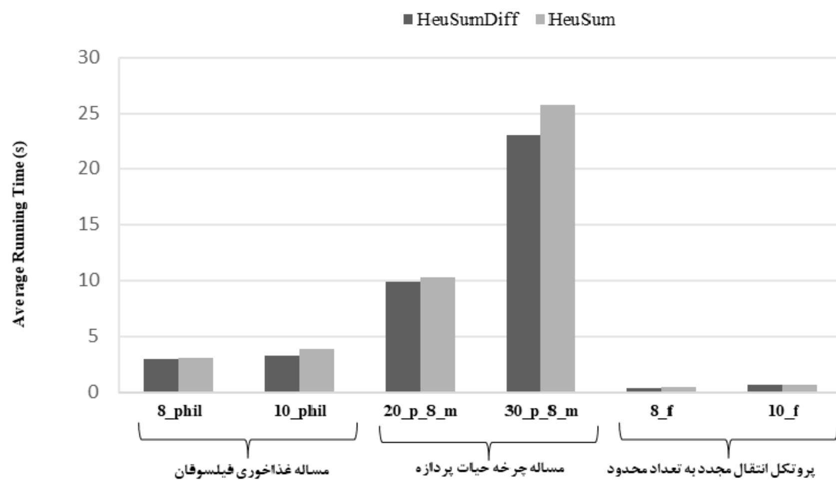
Archive of SID



شکل ۴: مقایسه میانگین زمان اجرای الگوریتم ژنتیک برای کشف بن‌بست.



شکل ۵: مقایسه میانگین زمان اجرای الگوریتم بهینه‌سازی ازدحام ذرات برای کشف بن‌بست.



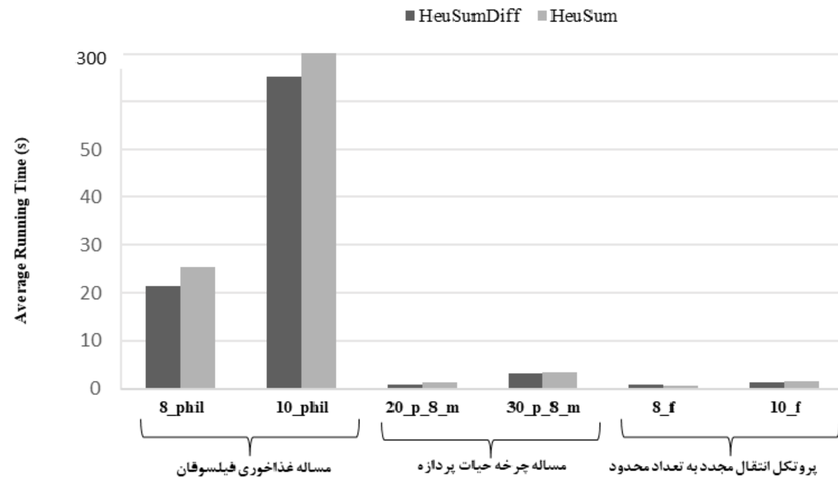
شکل ۶: مقایسه میانگین زمان اجرای الگوریتم جستجوی پرتو برای کشف بن‌بست.

۵- نتیجه‌گیری و کارهای آینده

ایمنی سیستم‌های نرم‌افزاری، خصوصاً از نوع بحرانی- ایمنی، باید قبل از پیاده‌سازی و در سطح مدل مورد تحلیل واقع شود. وارسی مدل، یک روش تحلیل دقیق ایمنی سیستم‌های نرم‌افزاری است که یک مدل به همراه ویژگی مورد نظر را دریافت کرده و با تولید تمام حالت‌های قابل دسترس، درستی ویژگی را بررسی می‌کند. محدودیت اصلی که این تکنیک ممکن است موقع وارسی سیستم‌های پیچیده و بزرگ با آن مواجه شود، کمبود حافظه در تولید همه حالت‌های ممکن (مشکل انفجار فضای حالت) است. عدم وجود حالت بن‌بست یکی از ویژگی‌های مهم در تحلیل ایمنی سیستم‌های نرم‌افزاری است که بررسی آن ممکن است با این مشکل مواجه شود. در این مقاله، یک هیوریستیک جدید برای یافتن بن‌بست در فضای حالت مدل ارائه گردید و سرعت تشخیص آن با به کار بردن در الگوریتم‌های جستجوی مکاشفه‌ای ساده از جمله عمقی تکرارشونده A* و جستجوی پرتو و الگوریتم‌های تکاملی مختلف از جمله ژنتیک، بهینه‌سازی ازدحام ذرات و بهینه‌سازی بیزی با کارایی

www.SID.ir

Archive of SID



شکل ۷: مقایسه میانگین زمان اجرای الگوریتم جستجوی عمقی تکرارشونده *A برای کشف بن‌بست.

هیوریستیک قبلی مقایسه گردید. مقایسه نتایج تجربی تأیید می‌کنند که هیوریستیک جدید می‌تواند حالت بن‌بست را در زمان کمتری نسبت به هیوریستیک قبلی پیدا کند. بهبود هیوریستیک ارائه‌شده در این مقاله می‌تواند به عنوان کار بعدی در نظر گرفته شود.

مراجع

[1] C. Baier and J. P. Katoen, *Principles of Model Checking*, MIT Press, 2008.

[2] J. C. Bicarregui and B. M. Matthews, "Proof and refutation in formal software development," in *Proc. 3rd Irish Workshop on Formal Methods*, pp. 1-15, Galway, Ireland, 1-2 Jul. 1399.

[3] R. Behjati, M. Sirjani, and M. Nili Ahmadabadi, "Bounded rational search for on-the-fly model checking of LTL properties," In: F. Arbab and M. Sirjani, (eds) *Fundamentals of Software Engineering. FSEN 2009. Lecture Notes in Computer Science*, vol 5961. Springer, Berlin, Heidelberg. pp. 292-307, 2009.

[4] S. Edelkamp, A. L. Lafuente, and S. Leue, "Protocol verification with heuristic search: first results," in *Proc. AAAI Sym. on Model-Based Validation of Intelligence*, pp. 75-83, Menlo Park, CA, USA, 9 Oct. 2000.

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

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

[7] R. Yousefian, V. Rafe, and M. Rahmani, "A heuristic solution for model checking graph transformation systems," *Applied Soft Computing*, vol. 24, pp. 169-180, Nov. 2014.

[8] E. Alba, F. Chicano, M. Ferreira, and J. Gomez-Pulido, "Finding deadlocks in large concurrent java programs using genetic algorithms," in *Proc. of the 10th Annual Conf. on Genetic and Evolutionary Computation*, pp. 1735-1742, Atlanta, GA, USA, 12-16 Jul. 2008.

[9] E. Alba and F. Chicano, "Finding safety errors with ACO," in *Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation*, pp. 1066-1073, London, England, 7-11 Jul. 2008.

[10] F. Chicano, M. Ferreira, and E. Alba, "Comparing metaheuristic algorithms for error detection in java programs," in *Proc. Int. Symp. on Search Based Software Engineering,* pp. 82-96, Szeged, Hungary, 10-12 Sept. 2011.

[11] V. Rafe, M. Moradi, R. Yousefian, and A. Nikanjam, "A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations," *Applied Soft Computing*, vol. 33, pp. 136-149, Aug. 2015.

[12] M. Ferreira, F. Chicano, E. Alba, and J. Gomez-Pulido, "Detecting protocol errors using particle swarm optimization with java pathfinder," in *Proc. of the High Performance Computing & Simulation Conf.,* pp. 319-325, Nicosia, Cyprus, 3-6 Jun. 2008.

[13] H. Kastenberg and A. Rensink, "Model checking dynamic states in GROOVE," in *Proc. Int. SPIN Workshop on Model Checking of Software*, pp. 299-305, Vienna, Austria, 30 Mar.-1 Apr. 2006.

[14] G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1997.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, Jul. 1968.

[16] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, John Wiley & Sons, 2004.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int Conf. on Neural Networks,* , vol. 4, pp. 1942-1948, Perth, Australia, 27 Nov.-1 Dec. 1995.

[18] P. Larranaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Springer Science & Business Media, 2001.

[19] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[20] A. Schmidt, "Model checking of visual modeling languages," in *Proc. Conf. of PhD Students in Computer Science*, p. 102, 2004.

[21] J. F. Groote and J. V. D. Pol, "A bounded retransmission protocol for large data packets," in *Proc. Int. Conf. on Algebraic Methodology and Software Technology,* pp. 536-550, Munich, Germany, 1-5 Jul. 1996.

[22] E. Pira, V. Rafe, and A. Nikanjam, "Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations," *Information and Software Technology*, vol. 97, pp. 110-134, May 2018.

عین‌الله پیرا در سال ۱۳۷۹ مدرک کارشناسی مهندسی کامپیوتر خود را از دانشگاه خوارزمی، در سال ۱۳۸۱ مدرک کارشناسی ارشد را از دانشگاه صنعتی شریف و در سال ۱۳۹۶ مدرک دکتری را از دانشگاه اراک دریافت نمود. دکتر پیرا از سال ۱۳۹۷ در دانشکده فناوری اطلاعات و مهندسی کامپیوتر دانشگاه شهیدمدنی آذربایجان در تبریز مشغول به فعالیت گردید و اینک نیز عضو هیأت علمی این دانشکده می‌باشد. زمینه‌های علمی مورد علاقه نام‌برده شامل موضوعاتی مانند مدل‌سازی و وارسی رسمی نرم‌افزار، محاسبات تکاملی، داده‌کاوی و یادگیری ماشین می‌باشد.

www.SID.ir