Gil, Víctor (2021). Revisión sistemática de literatura sobre generación automática de ayudas en ejercicios de programación. Cuaderno Activa, 13, 89-102.

Cuaderno Activa
•REVISTA CIENTÍFICA DE LA FACULTAD DE INGENIERÍA•

Revisión sistemática de literatura sobre generación automática de ayudas en ejercicios de programación

Víctor Daniel Gil Vera[1]

Recibido: 11 abril 2021. Aprobado: 9 mayo 2021.

Resumen: La programación de computadores es una de las habilidades más demandadas en el mercado laboral mundial y es un componente esencial del plan de estudios en cualquier programa universitario de ingeniería de sistemas. Una forma de ayudar a los estudiantes que tienen dificultades al momento de resolver los ejercicios es generar ayudas automáticas, las cuales consisten en suministrar sugerencias personalizadas durante el proceso de solución. Uno de los principales desafíos asociados con la generación de ayudas para la programación es la modelación automática de los pasos de la solución a partir de un gran número de soluciones correctas, debido a la diversidad de posibles soluciones que un estudiante puede escribir. El objetivo de este trabajo fue presentar una revisión sistemática de literatura sobre los algoritmos existentes para generar ayudas automáticas a partir de un conjunto de soluciones correctas. Se concluye que, a pesar de que diferentes investigaciones han demostrado la efectividad de este tipo de ayudas, su empleabilidad de manera masiva apenas comienza a implementarse en universidades de América Latina.

Palabras clave: Árboles de Sintaxis Abstracta, Inteligencia artificial, Programación de Computadores, Sistemas Tutoriados Inteligentes.

Abstract: Worldwide, computer programming is one of the most demanded skills in the labor market and is an essential component of the curriculum in any university systems engineering program. One way to help students who have difficulties in solving exercises is the generation of automatic hints, which consist of providing personalized hints during the solution process. One of the main challenges associated with the generation of programming hints is the automatic modeling of the solution steps from a large number of correct solutions, due to the diversity of possible solutions that a student can write. The objective of this paper was to present a systematic literature review of existing algorithms for generating automatic hints from a set of correct solutions. This paper concludes that, in spite of the fact that different researches have demonstrated the effectiveness of this type of hints, their massive employability is just beginning to be implemented in Latin American universities.

Keywords: Abstract Syntax Trees, Artificial Intelligence, Computer programming, Intelligent Tutorial Systems.

1 *Víctor Daniel Gil Vera. Doctor en Ingeniería de Sistemas. Universidad Católica Luis Amigó. Colombia, Medellín. victor.gilve@amigo.edu.co. ORCID:0000-0003-3895-4822.*

Introducción

La retroalimentación es un componente esencial del proceso de aprendizaje en cualquier área de conocimiento. El tamaño de las clases en las ciencias de la computación aumenta rápida y constantemente, lo cual hace difícil brindar retroalimentación a un gran número de estudiantes al mismo tiempo. A pesar de que los Sistemas Tutoriales Inteligentes (STI) proporcionan comentarios personalizados automáticamente, su construcción requiere tiempo y conocimiento experto, especialmente para generar sugerencias a los estudiantes que tienen dificultades en la resolución de un ejercicio [1]. Los enfoques basados en datos se pueden utilizar para proporcionar sugerencias personalizadas de forma automática y a escala mediante el uso de soluciones correctas.
Este trabajo presenta los resultados de una revisión sistemática de literatura (RSL), la cual tuvo como objetivo dar respuesta a preguntas de investigación a partir de los resultados publicados en investigaciones científicas. Para su elaboración, se han seguido las directrices de la declaración PRISMA [2], [3], en la cual se establecen los lineamientos para hacer una correcta RSL. Se realizó una búsqueda en las bases de datos científicas Scopus y WoS, se consideraron todas las publicaciones disponibles hasta octubre de 2021 con el objetivo de responder las siguientes preguntas de investigación:

- P1. ¿Cuáles son los enfoques actuales de la generación de ayudas basadas en datos para la escritura de algoritmos?
- P2. ¿Cuáles son los principales algoritmos para la generación de ayudas basadas en datos?
- P3. ¿Cuáles son las brechas de conocimiento de la generación de ayudas basadas en datos para la escritura de algoritmos?
- P4. ¿Qué STI existen en la actualidad para la enseñanza de la programación de computadores?

Las palabras claves empleadas para la selección de los documentos fueron: *"data guides"*, *"computer"*, *"science"*, *"education"*, *"programming"*, *"ITS" e "informatics"*.

Se empleó la siguiente cadena de búsqueda que incluía sinónimos comunes para la educación en programación y la tutoría inteligente:

TITLE-ABS-KEY ("computer science education" OR "data-driven" OR "hint" OR "next-step" OR "programming" OR "software engineering education" OR "introductory computer" OR "introductory programming" OR "teach-programming" OR "learn-programming" OR "novice-programming" OR "coding education" OR "introductory computer science") AND ("intelligent tutor*" OR "adaptive tutor*" OR "cognitive tutor*" OR "smart tutor")

Los criterios de inclusión y exclusión se diseñaron para extraer solamente los artículos que estuvieran relacionados con las preguntas de investigación planteadas. Para determinar si los artículos cumplían con los criterios de inclusión o exclusión, se leyeron los resúmenes de cada uno y se descartaron los que no. Los criterios de inclusión que se emplearon para seleccionar los artículos considerados fueron:

- Criterio 1: emplearon algoritmos para la generación de ayudas automáticas y fueron utilizados por estudiantes de cursos de programación de computadores.
- Criterio 2: los algoritmos se emplearon para la enseñanza de un lenguaje de programación que tiene estructuras de control condicionales e iterativas.
- Criterio 3: los algoritmos tomaban un conjunto de datos de entrenamiento de trazas de soluciones correctas y un conjunto de solicitudes de ayudas como entradas y devolvían un conjunto de sugerencias/pistas o ayudas.

Para descartar los artículos de la búsqueda primaria, los criterios de exclusión fueron:

- Criterio 1: no se evidenció que los algoritmos hayan sido empleados por estudiantes que aprenden programación de computadores.

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 91 de 130

- Criterio 2: los algoritmos no fueron validados.
- Criterio 3: los algoritmos no servían para generar ayudas automáticas.

De la RSL se identificó que EE. UU., Reino Unido y Australia son los tres países pioneros en esta área de conocimiento, todas las universidades de América Latina están llamadas a emplear este tipo de ayudas, ya que son grandes los beneficios que genera su uso en cursos universitarios de programación de computadores. Se concluye que la generación de ayudas automáticas basadas en datos es de gran utilidad para los docentes de esta área de conocimiento, sobre todo cuando es alto el número de estudiantes matriculados, ya que es complejo brindar asesoría personalizada a cada uno de manera independiente. A pesar de que los algoritmos existentes en la actualidad para generar este tipo de ayudas han demostrado tener un grado alto de efectividad, uno de los puntos débiles es la generación de sugerencias que no tienen relación con lo que están desarrollando los estudiantes, lo que los desmotiva a solicitar las ayudas. Esta problemática es una de las principales brechas de conocimiento que puede abordarse en futuras investigaciones.

Marco Teórico

Las habilidades de programación se están convirtiendo en una competencia central para casi todas las profesiones. Por lo anterior, la educación en programación (*JavaScript, Python, Java, TypeScript, C#, PhP, C++, C, Shell, Ruby, R, VBA, Swift, Kotlin*) cada vez más se integra en los planes de estudio de los programas de educación superior, especialmente en los relacionados con la ingeniería [4]. Las dificultades con el aprendizaje de la programación son una de las principales barreras que tienen los estudiantes para adelantar estudios en informática y en otras disciplinas relacionadas, lo cual obedece principalmente a su incapacidad para resolver los ejercicios, situación que puede desmotivarlos a avanzar cuando no pueden obtener ayuda de inmediato por parte del docente o tutor.

Para abordar esta problemática, se han propuesto varios enfoques para ayudar a los estudiantes a aprender a resolver ejercicios de programación. Tradicionalmente, la mejor opción en la enseñanza de la programación ha sido la tutoría "cara a cara" y "uno a uno". Sin embargo, los tutores humanos no siempre están disponibles y es complejo brindar asesoría personalizada a un número grande de estudiantes. La Tutoría Basada en Datos (TBD) es un subcampo de la tutoría inteligente (TI) que basa la toma de decisiones en el trabajo de antiguos estudiantes, en lugar de una base de conocimientos construida por expertos o un gráfico con todas las rutas posibles. El primer avance de la TBD con un enfoque en la generación de ayudas fue iniciado por la "Fábrica de Ayudas", un sistema para tutores lógicos que utiliza los procesos de decisión de Markov para integrar las rutas de acción de los estudiantes en un solo gráfico [5].

Las soluciones correctas que han realizado los estudiantes se pueden utilizar para proporcionar sugerencias personalizadas de forma automática. En lugar de construir una gran base de conocimiento de tutores expertos (BC), el enfoque basado en datos utiliza una gran cantidad de soluciones estudiantiles correctas. En resumen, la TBD emplea las soluciones correctas de los estudiantes para construir un espacio de solución que contenga todas las soluciones que los estudiantes han creado en semestres anteriores en un curso de programación, de manera tal que se puedan establecer diferentes caminos posibles para corregir intentos de los estudiantes de manera automática. En el campo de la programación, existen algoritmos que generan automáticamente ayudas que les sugieren a los estudiantes cómo deben editar su código para resolver errores y progresar, Se destacan los algoritmos: *TR-ER, CHF, NSNLS, CTD e ITAP*, los cuales han demostrado mejorar el aprendizaje de los estudiantes en esta área de conocimiento [4].

Planteamiento del Problema

Objetivo general

Realizar una RSL enfocada a determinar el estado actual de conocimiento sobre la generación de ayudas automáticas basadas en datos en cursos de programación de computadores.

Objetivos específicos

- Identificar los enfoques actuales de la generación de ayudas basadas en datos para la escritura de algoritmos.
- Identificar los principales algoritmos para la generación de ayudas basadas en datos.
- Identificar las brechas de conocimiento de la generación de ayudas basadas en datos para la escritura de algoritmos.
- Identificar los STI existentes en la actualidad para la enseñanza de la programación de computadores.

Pregunta de investigación

P1. ¿Cuál es el estado actual de conocimiento sobre la generación de ayudas automáticas basadas en datos en cursos de programación de computadores?

Justificación

Es necesario realizar una revisión exhaustiva de literatura enfocada a determinar el estado actual de conocimiento sobre la generación de ayudas automáticas basadas en datos en cursos de programación de computadores, ya que este tipo de cursos en la mayoría de universidades tienen la tendencia a crecer permanentemente, lo que significa un incremento constante en el número de estudiantes matriculados y un enorme desafío para los docentes, sobre todo al momento de atender las dudas e inquietudes que presenten los estudiantes cuando deben resolver los ejercicios [37]. Estas ayudas permiten mantener la motivación de los estudiantes, ya que de manera más fácil y eficiente obtienen sugerencias cuando se les presenten dificultades, sin tener que estar supeditados a la orientación del docente o tutor.

Metodología

Las RSL se caracterizan por estar enfocadas en responder preguntas de investigación claramente definidas y por emplear métodos sistemáticos explícitos para identificar, seleccionar y evaluar críticamente la investigación relevante de los estudios publicados con anterioridad relacionados con un tema por investigar; son una forma metódica para identificar, evaluar e interpretar los estudios empíricos disponibles sobre un tema, pregunta de investigación o fenómeno de interés [6].

La declaración PRISMA es un conjunto mínimo de elementos basados en evidencia científica para garantizar la calidad de las revisiones sistemáticas y meta-análisis [7], consta de una lista de verificación de 27 elementos y un diagrama de flujo de cuatro fases. La lista de verificación incluye elementos que se consideran esenciales para la presentación de informes transparentes de una revisión sistemática [8]. Esta declaración es útil para la evaluación crítica de revisiones sistemáticas publicadas previamente [9].

Aplicando la cadena de búsqueda presentada anteriormente, se obtuvieron 23 resultados en WoS y 26 en Scopus, para un total de 49 resultados. También se identificaron 3 publicaciones divulgativas relacionadas con la temática en otro tipo de fuentes. Tras eliminar seis duplicados entre las dos bases de datos, quedaron 46 artículos, a los cuales se les hizo el cribado con base en la lectura de los títulos y resúmenes, a partir de esta lectura se descartaron 7 artículos por no tener relación con cursos de programación.

De los 39 artículos que quedaron, y según los criterios de inclusión y exclusión mencionados más arriba, se descartaron 9 artículos porque: no existía evidencia de que los algoritmos que emplearon hayan sido utilizados por estudiantes que aprenden programación (n=2); eran algoritmos que no se habían validado (n=3), o no servían para generar ayudas automáticas (n=4). Finalmente, 30 artículos cumplieron los criterios de inclusión y se seleccionaron para llevar a cabo la revisión sistemática. La Figura 1 resume el proceso mediante un diagrama de Flujo PRISMA en cuatro niveles.

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 93 de 130

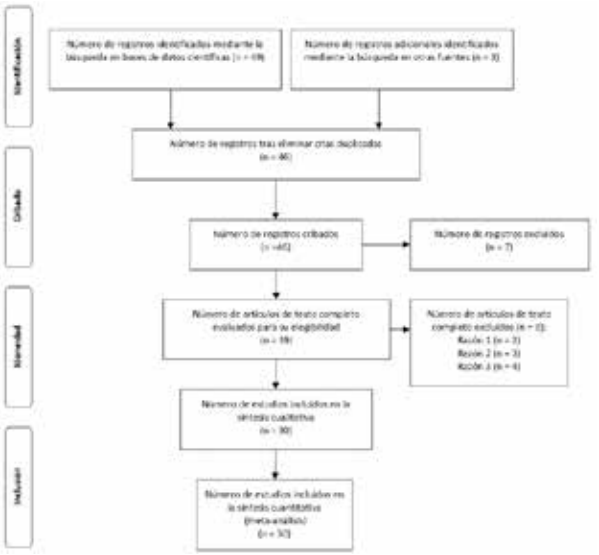

Figura 1. Diagrama de flujo en cuatro niveles.
Fuente: elaboración propia.

Se debe señalar que se verificó el cumplimiento de los 27 ítems de chequeo que establece la declaración: Título, Resumen estructurado, Introducción (Justificación y Objetivos), Métodos (Protocolo y Registro, Criterios de elegibilidad, Fuentes de información, Búsqueda, Selección de los estudios, Proceso de recopilación de datos, Lista de datos, Riesgo de sesgo en los estudios individuales, Medidas de resumen, Síntesis de resultados, Riesgo de sesgo entre los estudios, Análisis adicionales), Resultados (Selección de estudios, Características de los estudios, Riesgo de sesgo en los estudios, Resultados de los estudios individuales, Síntesis de los resultados, Riesgo de sesgo entre los estudios, Análisis adicionales), Discusión (Resumen de la evidencia, Limitaciones, Conclusiones) y Financiación.

Resultados y discusión

En esta sección se presentan los resultados encontrados en la revisión del estado del arte en cada una de las preguntas de investigación:

P1. ¿Cuáles son los enfoques actuales de la generación de sugerencias basadas en datos para la escritura de algoritmos?
A continuación, se detalla cada uno de los cinco enfoques existentes, los cuales se basan en la generación de sugerencias centradas en correcciones y en estilos de código [10], [11].

Enfoque de síntesis del programa
En [12] emplearon modelos de error y bocetos de programas para encontrar una asignación de estudiantes a programas actuales. En lugar de confiar en un conjunto predefinido de soluciones, emplearon la síntesis del programa para generar una nueva solución a partir del programa actual del estudiante. Sin embargo, de acuerdo con [13], este enfoque requiere expertos que definan un modelo de error específico para cada ejercicio de programación.

En la investigación desarrollada en [14], se basaron en el análisis de las ediciones de una sola línea realizadas por estudiantes, las cuales se emplearon para intentar encontrar una solución correcta. Esas ediciones se utilizaron como fuente para proporcionar sugerencias a los estudiantes. Sin embargo, su aplicación requiere un conjunto de casos de prueba para evaluar los códigos generados [15]. En [16], todas las expresiones comunes de los códigos de los estudiantes se emplearon para crear una base de datos de códigos fuente la cual fue utilizada para la generación de ayudas. Esta técnica tiene un gran potencial para apoyar soluciones nuevas, pero falla cuando el código del estudiante que solicita la ayuda difiere de las soluciones de la base de datos.

En [17] se adoptó un enfoque basado en ejemplos para aprender las correcciones de código y se emplearon transformaciones de árbol de sintaxis abstracta (AST) a partir de pares de envíos de estudiantes incorrectos y correctos. A pesar de que este enfoque requiere mucho menos esfuerzo, no genera sugerencias cuando el código del estudiante que solicita la ayuda se aleja de una solución correcta [14]. Por su parte, en [18] introdujeron un enfoque de iniciativa mixta que combina la experiencia del profesorado con técnicas de síntesis de programas basadas en datos. En esta investigación se demostró cómo el análisis y la síntesis de programas pueden usarse como ayuda para que un maestro pueda escalar la retroalimentación basado en su profundo conocimiento. Sin embargo, se requiere que los

docentes revisen y escriban sugerencias para códigos incorrectos de los estudiantes [15]. También exploraron un espacio de diseño de ayudas que puede generarse automáticamente a partir de las transformaciones de código aprendidas de la síntesis del programa. En esta investigación, las estrategias que emplea un docente humano fueron adaptadas a sugerencias automatizadas y se identificaron cinco tipos de ayudas para los docentes que pueden generarse por síntesis del programa: transformaciones, ubicaciones, datos, comportamiento y ejemplos [15].

En [14], desarrollaron un sistema robusto de generación de ayudas que extiende la cobertura del enfoque basado en síntesis del programa empleado, y un verificador de sintaxis que detecta errores comunes de sintaxis en subexpresiones individuales, para guiar a los estudiantes a soluciones parciales que puedan evaluarse para la exactitud semántica. El enfoque basado en la síntesis del programa se utiliza para generar ayudas para programas casi correctos, si el enfoque falla, el analizador de casos detecta ramas del programa que faltan para guiar a los estudiantes a soluciones parciales con estructuras razonables.

Enfoque basado en agrupaciones

En [16] emplearon la agrupación para inferir grupos de programas de computadora y seleccionar la solución de muestra más similar para generar ayudas. Cuando el estudiante requiere una sugerencia sobre cómo cambiar su código para acercarse a una solución correcta, se compara con un ejemplo similar del grupo, y las diferencias entre su código y el código de ejemplo se comparan para ayudar al estudiante a mejorar su propia solución. El desafío con este enfoque es la derivación de pasos de solución a partir de soluciones completas de muestra para reducir el esfuerzo en la modelación de ejemplos.

En [19] introdujeron una representación alternativa de programas de computadora para la clasificación y detección de errores en STI, es decir, trazas de ejecución. La representación de traza se puede aplicar para identificar programas erróneos, lo que permite que un STI detecte si un estudiante ha terminado una tarea o aún necesita continuar. Sin embargo, concluyeron que se requiere una representación sintáctica cuando un programa aún no compila o se bloquea.

En esta investigación también propusieron una técnica semisupervisada para generar retroalimentaciones. Esta técnica agrupa las soluciones basadas en estrategias. Los docentes etiquetan manualmente en cada grupo un intento correcto y validan formalmente las soluciones incorrectas contra la correcta. En [20] se presentó una técnica novedosa para agrupar y reparar ejercicios de programación. Los intentos correctos se agrupan empleando variables para diferentes entradas. Finalmente, se selecciona un intento de cada grupo como solución de referencia y se hacen las comparaciones de código. Un intento incorrecto se compara con cada intento correcto del conjunto de solución empleando rastros de variables para calcular reparaciones. Esta técnica proporciona comentarios personalizados utilizando la solución de referencia con el menor número de variables. Sin embargo, el problema de esta técnica es que requiere entradas que no son fáciles de proporcionar para activar todos los errores posibles [21].

Enfoque de recomendación

En [22] se representa un marco que puede ayudar a los estudiantes en el proceso de codificación, recomendando ediciones de códigos específicos relevantes para los códigos de solución. Se emplea el algoritmo *pq-Gram* para reducir las distancias entre los árboles de sintaxis abstracta (AST) y se compara la solución del estudiante con la solución más cercana en una base de datos de soluciones correctas para identificar el conjunto de inserciones, eliminaciones y reetiquetado que transformará directamente el AST del estudiante en una solución correcta. La desventaja de este método es que requiere obligatoriamente la generación de AST, la similitud semántica entre los códigos y las pruebas de usabilidad. En [23] se presenta un marco denominado Sistema de recomendación de ejemplo (ERS), que se basa en EBL y que utiliza algoritmos de minería de vanguardia para recomendar una lista enfocada, organizada y personalizada de ejemplos resueltos con el objetivo general de aumentar la probabilidad

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 95 de 130

de éxito de los estudiantes. La limitación de este sistema es la construcción manual de expresiones regulares (RE) por expertos.

Enfoque de razonamiento basado en casos

En [24] emplearon un enfoque de razonamiento basado en casos (CBR), al que llaman Recuperación de Árbol de Sintaxis Abstracta (ASTR), para extraer datos de soluciones anteriores contenidas en un gran conjunto de datos. Este sistema no requiere conocimiento previo del problema que se está resolviendo. Emplea CBR y la gramática del lenguaje de programación para recuperar una solución previa con alta similitud. Sin embargo, el sistema no contiene información sobre el problema de programación y no funciona para el lenguaje Python.

Sugerencias basadas en la "Fábrica de Ayudas"

Esta técnica representa las interacciones del estudiante con el docente en forma de gráfico. Cuando el estudiante solicita la ayuda y su código coincide con alguna parte del gráfico, se genera una ayuda que finalmente conduce a una solución [25]. En [26] la emplearon para generar automáticamente ayudas en un curso de lógica proposicional. Este enfoque utiliza los datos del estudiante para construir un proceso de decisión de Markov, opera representando el problema en un gráfico con nodos, donde cada nodo representa el estado actual del estudiante en la resolución del problema y cada línea, las acciones del estudiante. Una solución se representa como una ruta desde el estado inicial a un estado objetivo. Este enfoque se ha extendido para trabajar en otros dominios más estrechamente relacionados con la programación. En [27] lo emplearon en un tutor llamado *iList*, que ayuda a los estudiantes a aprender listas vinculadas, y en [28] lo emplearon para representar estados de los programas que desarrollaban los estudiantes. En [29] señalaron que múltiples soluciones que existen para estudiantes deberían estar disponibles, debido al riesgo de que no se reconozca una alternativa específica para resolver el ejercicio. En [1] propusieron un enfoque basado en datos para crear un espacio de solución consistente con todas las rutas posibles desde el enunciado del problema hasta una solución correcta. Estos investigadores desarrollaron ITAP, un asistente de enseñanza inteligente para programación, el cual permite generar ayudas para estados nunca antes considerados.

Según [22], un obstáculo importante de las representaciones con AST es la pérdida de información de comportamiento. En [26], presentaron un nuevo algoritmo basado en datos llamado *Contextual Tree Decomposition* (CTD), basado en "Fábrica de Ayudas", para generar ayudas para estos programas. En [29] desarrollaron *iSnap*, una extensión del entorno de programación *Snap* que agrega características clave de los STI, incluido el registro detallado y sugerencias generadas automáticamente.

La Tabla 1 presenta las 10 publicaciones con la mayor cantidad de citas en orden descendente y la principal contribución de cada una.

Tabla 1. Publicaciones más citadas y su contribución.

Autores y año	Citas	Contribución
[1]	87	La abstracción de estados, la construcción de rutas y la cosificación de ayudas se pueden emplear de diferentes maneras para brindar a los estudiantes diferentes tipos de retroalimentación; existe un gran potencial en el uso de funciones para identificar códigos ineficientes y mejorar su estilo y eficiencia.
[26]	67	Los entornos de programación diseñados intencionalmente para ayudar a los estudiantes principiantes se han vuelto cada vez más populares; a pesar de que estos entornos ofrezcan herramientas para reducir errores de sintaxis, la mayoría de ellos ofrecen poca ayuda a los estudiantes que se atascan y necesitan apoyo de expertos.
[18]	50	Las funciones de autocorrección de sintaxis de código tienen la capacidad de identificar errores cometidos por los estudiantes, pero carecen del conocimiento experto de un docente, esto conlleva la generación de correcciones funcionalmente correctas, pero estilísticamente pobres.

Autores y año	Citas	Contribución
[30]	28	Los STI han demostrado tener éxito en el dominio de la programación al proporcionar sugerencias y comentarios personalizados a los estudiantes; sin embargo, muchos entornos de programación populares para principiantes carecen de estas funciones inteligentes.
[28]	27	La generación de ayudas automáticas es una tarea que requiere mucho tiempo para poder ser implementadas satisfactoriamente en STI; sin embargo, las técnicas de minería de datos educativos (EDM) y de aprendizaje automático (ML) pueden facilitar esta tarea.
[31]	25	La calidad de las primeras ayudas o sugerencias suministradas a los estudiantes en la resolución de un ejercicio de programación los motiva o desmotiva a seguirlas solicitando, razón por la cual se deben presentar ayudas útiles que tengan relación con lo que han realizado los estudiantes para que puedan culminar los ejercicios de manera satisfactoria.
[32]	24	A medida que más entornos de programación agregan funciones de registro y los datos de programación se vuelven más accesibles, es esencial tener en cuenta cómo se comparten y usan estos datos. Para generar ayudas automáticas es importante identificar qué datos son importantes recopilar, dónde se pueden recopilar y cómo manejar la privacidad y el anonimato de los estudiantes.
[22]	22	Para resolver problemas que puedan presentar los estudiantes al momento de resolver un ejercicio de programación, se pueden emplear conjuntos de ediciones textuales de estudiantes de cursos pasados con el fin de sintetizar nuevos códigos aplicando secuencias de ediciones hasta identificar una solución al problema.
[33]	19	Para generar comentarios inteligentes en forma de ayudas personalizadas, se pueden emplear métodos basados en el uso de AST que representen los códigos de lo que llevan desarrollado los estudiantes y los comparen con otros AST culminados de manera satisfactoria por otros estudiantes.
[34]	7	Los STI basados en datos aprenden a proporcionar comentarios basados en el comportamiento de estudiantes de cursos pasados, lo que reduce el esfuerzo requerido para su desarrollo. Un obstáculo importante para la aplicación de métodos basados en datos en el dominio de la programación es la falta de acciones observables significativas para describir el proceso de resolución de problemas de los estudiantes.

Fuente: elaboración propia.

P2. ¿Cuáles son los principales algoritmos para la generación de ayudas basadas en datos?

Existen seis algoritmos de generación de ayudas automáticas basadas en datos: *TR-ER, CHF, NSNLS, CTD, SourceCheck* e *ITAP*. Cada algoritmo toma un conjunto de datos de entrenamiento de trazas de soluciones correctas y un conjunto de solicitudes de sugerencias como entradas, y devuelve un conjunto de correcciones correctas (sugerencias de respuesta). Cada ayuda generada se representa como un estado: el nuevo estado de código que resulta de aplicar la edición recomendada al estado de código actual del estudiante.

La mayoría de estos algoritmos funcionan en dos fases: primero identifican un estado de código objetivo (a menudo una solución correcta) en el conjunto de datos de capacitación, luego recomiendan ediciones para convertir el código del estudiante a este estado objetivo. En la Tabla 2 se describe cada uno de ellos.

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 97 de 130

Tabla 2. Descripción de los algoritmos.

Algoritmo	Descripción
TR-ER	Este algoritmo define el estado objetivo como la solución correcta más cercana al código actual del estudiante usando la fórmula de distancia pq-Gram para Árboles de Sintaxis Abstracta (AST) [35]; el cálculo de esta distancia implica representar un AST como un conjunto múltiple de todos sus subárboles. El algoritmo sugiere ayudas para insertar, eliminar o reemplazar código basado en los pq-Grams extras o faltantes en el código del estudiante [22].
CHF	Este algoritmo define el estado objetivo como el paso que un estudiante de cursos pasados haya realizado de manera acertada. Primero, calcula la distancia entre el rastro del estudiante, incluido el historial del código y los rastros de todos los estudiantes en el conjunto de datos de soluciones correctas que se acercaron a la solución, utilizando una deformación dinámica del tiempo (DDT). Segundo, define el estado objetivo basado en predicciones de cómo estos estudiantes procedieron en la situación del estudiante que está solicitando la ayuda, utilizando regresión del proceso gaussiano. Por último, identifica las ediciones de los AST que acercan más al estudiante al estado objetivo, utilizando una gramática de árbol para seleccionar solo ediciones válidas [35]
CTD	Este algoritmo, en lugar de seleccionar un único estado objetivo del conjunto de datos de solución, como hace la "Fábrica de Ayudas", descompone los AST de los estudiantes en subárboles y hace coincidir piezas más pequeñas de código con los de otros estudiantes y utiliza estas coincidencias para generar ayudas contextualizadas en cada subárbol [30].
Source Check	Este algoritmo selecciona un único estado objetivo, definido como la solución presentada más cercana al código actual del estudiante, lo calcula utilizando una métrica de distancia específica del código e incorpora elementos de código comunes de dos AST. El algoritmo genera sugerencias en cada nivel del AST basándose en las diferencias entre los nodos de dos AST [36].
ITAP	Este algoritmo genera ayudas mediante un proceso de cinco pasos: primero, elimina las variaciones sintácticas; segundo, identifica la solución correcta más cercana al código actual del estudiante; tercero, aplica la construcción de rutas para identificar cualquier solución correcta más cercana y no descubierta; cuarto, identifica un estado objetivo en la ruta hacia la solución óptima, y quinto, extrae la edición más cercana al nodo raíz del AST y la presenta al estudiante como una ayuda [1].
NSNLS	Este algoritmo determina el estado objetivo del código actual del estudiante que solicita la ayuda a partir de la solución del conjunto total de soluciones correctas que más se asemeja a lo que lleva desarrollado el estudiante, y genera como sugerencia el paso siguiente que se haya realizado en la solución identificada como la más similar del conjunto total de soluciones [22].

Fuente: elaboración propia.

P3. ¿Cuáles son las brechas de conocimiento de la generación de ayudas basadas en datos para STI en programación?

En el contexto de los STI basados en datos para la escritura de código, a pesar de los esfuerzos de investigación en los últimos años, la generación de ayudas automáticas sigue teniendo problemas. A partir de la revisión del estado del arte, las brechas que se identificaron fueron:

Modelado automático de los pasos de la solución de ejercicios a partir de soluciones correctas: ninguno de los STI para escritura de código modela automáticamente los pasos de la solución correcta de un conjunto de soluciones correctas de un ejercicio de programación. Cómo modelar automáticamente los pasos de la solución de una gran cantidad de soluciones correctas de un ejercicio de programación es un problema no resuelto.

***Lenguaje de programación*:** en el contexto de los STI basados en datos para la escritura de código, se puede ver que, aunque los STI que se han desarrollado previamente cubren muchos dominios, ninguno de ellos enseña programación en lenguaje C/C++.

Ejercicios de programación soportados por la escritura de códigos de STI basados en datos: es importante que un STI basado en datos para la escritura de código proporcione una colección de ejercicios de programación. Sin embargo, estos ejercicios de programación generalmente se almacenan en sistemas privados de las universidades.

Representación del código actual del estudiante: en el contexto de la "Fábrica de Ayudas", para poder generar automáticamente sugerencias, el estado actual de avance en la resolución de un problema se realiza a través de la captura del código actual del estudiante y no se considera su historial.

Similitud semántica: no han resuelto el problema de cómo extraer soluciones distintas de un gran conjunto de datos de manera eficiente y precisa.

P4. ¿Qué STI existen en la actualidad para la enseñanza de la programación de computadores?

De la búsqueda de STI para la programación se identificaron las siguientes variables: nombre, lenguaje de programación y características adaptativas primarias. Los criterios de presencia o ausencia se detallan a continuación:

I) Preguntas que los estudiantes responden dentro del STI
II) Creación de planes de programa por parte de los estudiantes con el STI
III) Generación de planes visuales de programas de enseñanza utilizados como recursos
IV) Lecciones sobre conceptos de programación
V) Materiales de referencia
VI) Soluciones trabajadas suministradas como un recurso de instrucción

La Tabla 3 presenta el resumen de los STI que fueron encontrados en la revisión.

Tabla 3. STI para la enseñanza de la programación

Nombre	Lenguaje	Característica adaptativa	I II III IV V VI
PROUST	Pascal	Retroalimentación	I, II, III, IV
LISP Tutor	Lisp	Retroalimentación	I, II, IV, V
ITEM/IP	Turingal	Retroalimentación y navegación	IV, V, VI
C-Tutor	C++	Retroalimentación	I, II, III, VI
ELM-ART	Lisp	Retroalimentación y navegación	V, VI
Scope Tutor	Pascal	Retroalimentación	I, II, III
ILMDA	Java	Retroalimentación	I, II, III, IV
AtoL	Java	Retroalimentación	V, VI
CIMEL ITS	Java	Retroalimentación y navegación	IV, V, VI
CPP-Tutor	C++	Retroalimentación y navegación	I, II, III
J-LATTE	Java	Retroalimentación	I, II, III, IV
ChiQat	No especifica	Retroalimentación	I, II, III
Ask-Elle	Haskell	Retroalimentación	I, II
ITAP	Python	Retroalimentación	IV, V, VI

Fuente: elaboración propia

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 99 de 130

En resumen, hay dos líneas de investigación propuestas para generar sugerencias basadas en datos en STI para la escritura de código: programas basados en síntesis y sugerencia basada en "Fábrica de Ayudas". Sin embargo, hay dos inconvenientes principales de los enfoques basados en síntesis: el primero hace referencia a que un instructor debe proporcionar manualmente modelos de error para cada problema, el segundo, a la escalabilidad, especialmente con programas grandes. En términos de conocimiento experto, los enfoques basados en "Fábrica de Ayudas" son adecuados para generar ayudas en STI para la escritura de códigos. Estos enfoques solo requieren dos conocimientos expertos para ejecutarse de forma independiente. Los STI existentes para la escritura de códigos que se basan únicamente en la generación de sugerencias de datos difieren entre sí en los siguientes aspectos: la representación del código actual del estudiante, la extracción de soluciones diferentes de un ejercicio de programación, el nivel de granularidad del código utilizado, el modelado automático de los pasos de la solución y el lenguaje de programación.

Conclusiones

En este trabajo se presentan los resultados de una RSL en la cual se identificaron los problemas, retos, vacíos y brechas en el área de la generación automática de ayudas de ejercicios de programación. Si se considera que el número de estudiantes en este tipo de cursos (Massive Open Online Courses —MOOC—) se incrementa constantemente, estos enfoques son de gran utilidad porque facilitan el proceso de retroalimentación y evaluación a los docentes de estos cursos. A pesar de que las limitaciones y desventajas de estos enfoques pueden desmotivar a los estudiantes, son más las ventajas y beneficios que generan, razón por la cual todos los programas de ingeniería que dicten cursos de programación en universidades de América Latina están llamados a implementarlos.

La mayoría de programas y plataformas en línea para la enseñanza de la programación (Google Colaboratory, SoloLearn, Learn Python, Programiz, Programming Hub, Enki, DataCamp, Mimo, etc.) tienen la capacidad de identificar errores de sintaxis en los códigos que desarrollan los estudiantes, y a pesar de que esta información es de gran importancia, en muchos casos no es suficiente, sobre todo cuando los estudiantes se enfrentan a ejercicios con un alto nivel de complejidad, lo que los obliga a consultar ayudas en repositorios como StackOverflow, GitHub, Medium, GeeksforGeeks, entre otros. Las ayudas que se almacenan en estos repositorios son de gran utilidad, sin embargo, los estudiantes pueden verse tentados a copiar código de ellos y pegarlo, lo que perjudica la adquisición y desarrollo de competencias de aprendizaje de manera autónoma.

Trabajo futuro

Futuras investigaciones pueden enfocarse en desarrollar aplicativos para generar ayudas automáticas en MOOC que se dictan en sistemas de gestión de aprendizaje como Moodle, Canvas, Chamilo, etc.

Financiación

Este trabajo fue patrocinado por la Universidad Católica Luis Amigó y fue desarrollado dentro del proyecto de investigación titulado: "Generación de ayudas automáticas en cursos de programación a partir de soluciones correctas" del grupo de investigación SISCO.

Referencias

[1] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," *Int. J. Artif. Intell. Educ.*, vol. 27, n.° 1, pp. 37-64, 2017.

[2] K. Knobloch, U. Yoon, and P. M. Vogt, "Preferred reporting items for systematic reviews and meta-analyses (PRISMA) statement and publication bias," *J. Cranio-Maxillofacial Surg.*, vol. 39, n.° 2, pp. 91-92, 2011.

[3] G. Urrutia y X. Bonfill, "La Declaración PRISMA: un paso adelante en la mejora de

las publicaciones de la Revista Española de Salud Pública," *Rev. Esp. Salud Pública,* vol. 87, n.° 2, pp. 99-102, 2013.

[4] N.-T. Le, "Analysis techniques for feedback-based educational systems for programming," in *Advanced Computational Methods for Knowledge Engineering, Springer,* T. B. Nguyen, T. van Do, H. A. Le Thi, N. T. Nguyen, Eds. Proceedings of the 4th International Conference on Computer Science, Applied Mathematics and Applications, ICCSAMA, 2-3 May, 2016, Vienna, Austria, pp. 141-152.

[5] T. Barnes and J. Stamper, "Toward automatic hint generation for logic proof tutoring using historical student data," in *International Conference on Intelligent Tutoring Systems,* 2008, pp. 373-382.

[6] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Inf. Softw. Technol.,* vol. 51, n.° 1, pp. 7-15, Jan. 2009.

[7] A. Liberati et al., "The PRISMA statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: explanation and elaboration," *J. Clin. Epidemiol.,* vol. 62, n.° 10, pp. e1-e34, 2009.

[8] J. Oláh, E. Krisán, A. Kiss, Z. Lakner, and J. Popp, "PRISMA Statement for Reporting Literature Searches in Systematic Reviews of the Bioethanol Sector," *Energies,* vol. 13, n.° 9, p. 2323, 2020.

[9] J. P. M. Peters, L. Hooft, W. Grolman, and I. Stegeman, "Reporting quality of systematic reviews and meta-analyses of otorhinolaryngologic articles based on the PRISMA statement," *PLoS One,* vol. 10, n.° 8, p. e0136540, 2015.

[10] R. R. Choudhury, H. Yin, and A. Fox, "Scale-driven automatic hint generation for coding style," in *International Conference on Intelligent Tutoring Systems,* 2016, pp. 122-132.

[11] E. S. Wiese, M. Yen, A. Chen, L. A. Santos, and A. Fox, "Teaching students to recognize and implement good coding style," in *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale,* 2017, pp. 41-50.

[12] R. Singh, "Accessible programming using program synthesis," Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, 2014.

[13] S. Terman, "GroverCode: code canonicalization and clustering applied to grading," Master of Engineering Thesis, Massachusetts Institute of Technology, 2016.

[14] P. M. Phothilimthana and S. Sridhara, "High-coverage hint generation for massive courses: Do automated hints help CS1 students?," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education,* 2017, pp. 182-187. https://doi.org/10.1145/3059009.3059058

[15] R. Suzuki, G. Soares, E. Glassman, A. Head, L. D'Antoni, and B. Hartmann, "Exploring the design space of automatically synthesized hints for introductory programming assignments," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems,* 2017, pp. 2951-2958.

[16] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart, "How to select an example? a comparison of selection strategies in example-based learning," in *International Conference on Intelligent Tutoring Systems,* 2014, pp. 340-347.

[17] R. Rolim et al., "Learning syntactic program transformations from examples," in *ICSE'17: Proceedings of the 39th*

ISSN: 2027-8101 - e-ISSN: 2619-5232. Enero-diciembre 2021 • página 101 de 130

International Conference on Software Engineering, 2017, pp. 404-415.

[18] A. Head et al., "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale,* 2017, pp. 89-98.

[19] B. Paaßen, J. Jensen, and B. Hammer, "Execution Traces as a Powerful Data Representation for Intelligent Tutoring Systems for Programming," in *Proceedings of the 9th Int. Educ. Data Min. Soc.,* 2016, pp. 183-190.

[20] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," *ACM SIGPLAN Not.,* vol. 53, n.° 4, pp. 465-480, 2018.

[21] V. J. Marin, T. Pereira, S. Sridharan, and C. R. Rivero, "Automated personalized feedback in introductory Java programming MOOCs," in *IEEE 33rd International Conference on Data Engineering (ICDE),* 2017, pp. 1259-1270.

[22] K. Zimmerman and C. R. Rupakheti, "An automated framework for recommending program elements to novices (n)," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 2015, pp. 283-288.

[23] R. Chaturvedi, "Task-based Example Miner for Intelligent Tutoring Systems," Ph. D. Thesis, University of Windsor, 2016.

[24] P. Freeman, I. Watson, and P. Denny, "Inferring student coding goals using abstract syntax trees," in *International Conference on Case-Based Reasoning,* 2016, pp. 139-153.

[25] M. T. Irfan and V. N. Gudivada, "Cognitive Computing Applications in Education and Learning," *Handbook of Statistics,* vol. 35, pp. 283-300, 2016.

[26] T. W. Price et al., "A Comparison of the Quality of Data-Driven Programming Hint Generation Algorithms," *Int. J. Artif. Intell. Educ.,* vol. 29, n.° 3, pp. 368-395, 2019.

[27] D. Fossati, B. Di Eugenio, S. Ohlsson, C. Brown, and L. Chen, "Data driven automatic feedback generation in the iList intelligent tutoring system," *Technol. Instr. Cogn. Learn.,* vol. 10, n.° 1, pp. 5-26, 2015.

[28] W. Jin, T. Barnes, J. Stamper, M. J. Eagle, M. W. Johnson, and L. Lehmann, "Program representation for automatic hint generation for a data-driven novice programming tutor," in *International Conference on Intelligent Tutoring Systems,* 2012, pp. 304-309.

[29] T. W. Price, Y. Dong, and D. Lipovac, "iSnap: towards intelligent tutoring in novice programming environments," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education,* 2017, pp. 483-488.

[30] T. W. Price, Y. Dong, and T. Barnes, "Generating Data-Driven Hints for Open-Ended Programming," *Int. Educ. Data Min. Soc.,* 2016.

[31] T. W. Price et al., "ProgSnap2: A Flexible Format for Programming Process Data," *Companion Proc. Int. Conf. Learn. Anal. Knowl. (LAK 2019),* pp. 1-7, 2019.

[32] T. W. Price, N. C. C. Brown, C. Piech, and K. Rivers, "Sharing and using programming log data," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education,* 2017, p. 729.

[33] A. Hicks, B. Peddycord, and T. Barnes, "Building games to learn from their players: Generating hints in a serious game," in *International Conference on Intelligent Tutoring Systems,* 2014, pp. 312-317.

[34] T. Lazar and I. Bratko, "Data-driven program synthesis for hint generation in programming tutors," in *International Conference on Intelligent Tutoring Systems,* 2014, pp. 306-311.

[35] B. Paassen, B. Mokbel, and B. Hammer, "Adaptive structure metrics for automated feedback provision in intelligent tutoring systems," *Neurocomputing,* vol. 192, pp. 3-13, 2016.

[36] M. Andrzejewska and A. Skawińska, "Examining students' cognitive effort during program comprehension – An eye tracking approach", *Artificial Intelligence in Education, AIED 2020,* Springer International Publishing, pp. 25-30, 2020.

[37] V. D. Gil Vera, "Análisis del aprendizaje: una revisión sistemática de literatura", *Cuad. Activa,* vol. 10, n.° 1, pp. 15-26, 2018.