Asistente Virtual para la Evaluación de Competencias Técnicas

A. Aguilar Cornejo[#1], M. Díaz Rodríguez[#2], L. Ramos Corchado[#3], M. Bernal Marín[#4],
N. Ruiz Monroy[#5], J. Gómez González[#6], A. Maytorena Ramírez[#7]
[#]Tecnológico Nacional de México: Instituto Tecnológico José Mario Molina Pasquel y Henríquez, Unidad Académica Zapopan, México. Ingeniería Electrónica. Departamento de Investigación.
[1]alejandro.aguilar@zapopan.tecmm.edu.mx, [2]miriam.diaz@zapopan.tecmm.edu.mx,
[3]leon.ramos@zapopan.tecmm.edu.mx, [4]miguel.bernal@zapopan.tecmm.edu.mx,
[5]nancy.ruiz@zapopan.tecmm.edu.mx, [6]za16011064@zapopan.tecmm.edu.mx,
[7]za16011491@zapopan.tecmm.edu.mx

***Resumen*—Este artículo presenta una plataforma que apoya la evaluación de competencias técnicas cuya implementación permite la generación y evaluación automática de ejercicios matemáticos en una dinámica académica que proporciona recursos de apoyo en materias de fundamentos computacionales.**

***Palabras clave*—fórmulas, generadores automáticos, intérpretes, asistente.**

I. Introducción

En la actualidad la educación requiere diversas transformaciones para adaptarse a las necesidades de los alumnos, nuevos modelos educativos son probados con el objetivo de lograr transmitir el conocimiento en las diversas áreas, diversas propuestas se muestran en [1-4].

Vivimos en una época en la que las Tecnología de la Información y la Comunicación (TIC) están integradas cada vez más en todas las actividades de la educación, de acuerdo con los nuevos modelos educativos. El día de hoy es común enviar y recibir información mediante canales virtuales haciendo uso de una computadora o de aplicaciones móviles de mensajería. En este contexto, el surgimiento de las plataformas educativas ha constituido un avance importante. Sitios como Edmodo, Schoology y Google Classroom permiten generar aulas virtuales, haciendo posible reunir en un solo sitio todos estos beneficios, además del acceso a recursos educativos, de evaluación y de seguimiento del curso. Algunos cursos se ofrecen, incluso, de manera virtual a través de la publicación de MOOC (Massive Open Online Curses) en sitios como edX, Coursera, Udacity, Khan Academy, Udemy y las plataformas virtuales de las universidades de Stanford, Harvard y Yale, entre muchas más.

Por otro lado, la atención personalizada en el aula es compleja y exige mucho tiempo y esfuerzo por parte del docente, particularmente cuando se trata de proveer y evaluar la resolución de ejercicios. Esto ocurre de manera específica y natural en las materias de contenido abstracto, en las cuales es necesario que el estudiante se incorpore frecuentemente a actividades de práctica. Por ejemplo, en el área de la computación se encuentran la lógica, la matemática discreta, la teoría de la computación y el modelado formal de procesadores de lenguajes.

Dadas las ventajas que proveen el uso de las TIC (Tecnologías de la Información y la Comunicación) y la automatización de estas tareas, se propone el desarrollo de esta plataforma que permita a los estudiantes contar con una fuente virtualmente inagotable de ejercicios de complejidad configurable y que esté disponible en todo momento, y a los profesores generar recursos, evidencias y resultados de evaluación de una manera completamente automática. Se prevé que esta herramienta sea desarrollada en distintas etapas. En este artículo se expone la primera de ellas, en la cual se desarrollará la plataforma con un sistema de cuentas de usuario y generación de cursos que se pondrá en línea, y se proveerá una colección inicial de algoritmos sobre tópicos selectos de matemáticas discretas. Subsecuentemente se irán incorporando nuevos módulos con motores dedicados para diversos tipos de problemas. De manera consecuente, la publicación de esta herramienta colocará al instituto en la vanguardia de la generación de tecnologías de la educación.

II. Matefácil Versión 2.0

El evaluador de competencias técnicas Matefácil es una herramienta que busca facilitar y agilizar la labor docente y los procesos de aprendizaje de los estudiantes, mediante la generación y evaluación automática de ejercicios matemáticos. En el 2020 se desarrolló la versión 1.0, en esta se enfocó en el desarrollo de la plataforma web y las tecnologías a utilizar, que permitió la prueba del funcionamiento general de la arquitectura propuesta. En esta versión 2.0 ya se plantea la solución integral con mejoras en diseño, experiencia de usuario y desempeño de la plataforma y la integración de la librería MathLive. La implementación se desarrolló en Django y Python, con su base de datos en PostgreSQL. La aplicación web y cada instancia de generador y solucionador está montada en contenedores (*dockers*).

A. Diseño de la Plataforma

En la fig. 1 se muestra el diseño Model Template View (MTV), que es el modelo de diseño utilizado para esta

aplicación. La capa de modelo se relaciona con la capa de acceso a datos. En esta primera capa se resuelve el acceso los datos: cómo acceder a ellos, cómo verificarlos, qué comportamiento tiene y qué relación existe entre los datos. En el nivel de presentación se resuelve la manera en que se muestra el contenido en páginas web y otros tipos de documentos. Finalmente, en la capa de vista se resuelve la lógica para acceder a los procesos y proporciona al usuario el modelo HTML correspondiente a pedido.

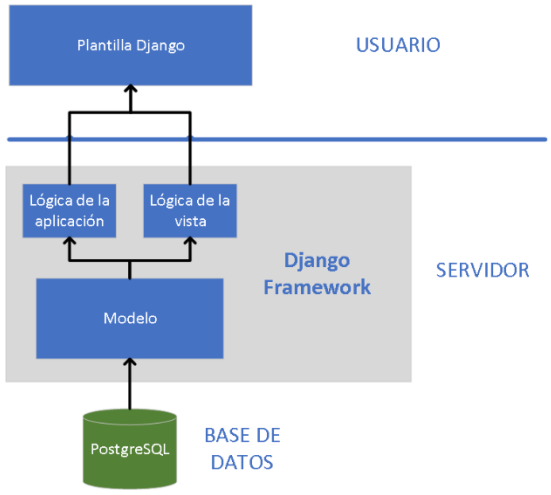

Fig. 1. Diseño la plaforma Matefácil.

B. Funcionalidad

Todas las vistas de la aplicación están generadas con el motor de Plantillas Django. La aplicación ofrece tres interfaces para los diferentes tipos de usuario: Estudiante, Maestro y Administrador. Se han habilitados las interfaces para las clases; un Maestro puede crear una clase y desde éstas pueden enviar actividades los Estudiantes que se hayan incorporado. Estas actividades pueden ser Tareas o Exámenes de 1 a 10 ejercicios cada una. Cuando el estudiante haya contestado la actividad, cada respuesta será evaluada y su resultado se les hará llegar a su interfaz correspondiente. En la fig. 2 se muestra la interfaz para estudiantes. En ella recibirán la retroalimentación de los ejercicios de práctica que le fueron asignados y contestados.

En la interfaz de Administrador se permite agregar, editar y eliminar contenido en la plataforma. La interfaz de administración lee los metadatos del modelo de la aplicación para proporcionar un entorno de actualización del producto que los usuarios del sitio pueden usar al instante.

III. Caso de estudio. Formulas proposicionales.

Para probar la funcionalidad de la plataforma se comenzó con ejercicios de lógica, con la generación aleatoria de ejercicios, y el diseño de un evaluador, considerando las reglas para el lenguaje formal planteadas en la siguiente sección.

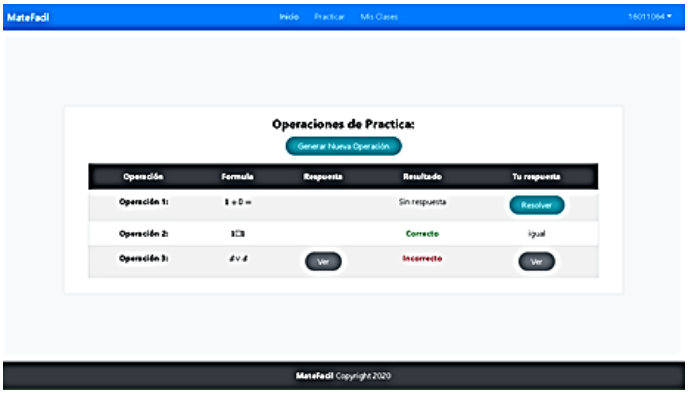

Fig. 2. Inferfaz de estudiante con la interfaz la plaforma Matefácil.

A. Sintaxis.

Para la explicación de este método se tomará como referencia el lenguaje (simplificado) de las fórmulas proposicionales. Para definirlo, asumimos el conjunto de *variables proposicionales* $\mathcal{P} = \{A, B, \dots, Z\}$ con dominio en el conjunto $\mathcal{B} = \{0,1\}$.

- Todo $\phi \in P$ es una fórmula proposicional.
- Si φ es una fórmula proposicional, entonces también $\neg\varphi$ es una fórmula proposicional. A $\neg\varphi$ le llamamos *la negación de* φ.
- Si φ y ψ son fómrulas proposicionales, entonces también $(\varphi \vee \psi)$ y $(\varphi \wedge \psi)$ son fórmulas proposicionales. A la fórmula $(\varphi \vee \psi)$ le llamamos *la disyunción* de φ y ψ, mientras que a $(\varphi \wedge \psi)$ le llamamos su *conjunción*.

Para la fórmula proposicional φ definimos su *soporte* como el conjunto de todas las variables que aparecen en ella y lo vamos a denotar $supp(\varphi)$. Por ejemplo, si $\varphi = (A \wedge (B \vee \neg C))$, entonces $supp(\varphi) = \{A, B, C\}$. Toda fórmula proposicional φ tiene un árbol de sintáxis $T(\varphi)$, que se define de la siguiente manera:

$$T(\varphi) = \begin{cases} \langle \vee, T(\varphi_1), T(\varphi_2) \rangle & \varphi = (\varphi_1 \vee \varphi_2) \\ \langle \wedge, T(\varphi_1), T(\varphi_2) \rangle & \varphi = (\varphi_1 \wedge \varphi_2) \\ \langle \neg, T(\varphi_1) \rangle & \varphi = \neg\varphi_1 \\ \langle \phi \rangle & \varphi = \phi, \phi \in \mathcal{P} \end{cases}$$

Por ejemplo, la fórmula $\varphi = (A \wedge (\neg P \vee Q))$ tiene el árbol que se muestra en la fig. 3.

$$\begin{aligned} T(\varphi) &= \langle \wedge, T(A), T(\neg P \vee Q) \rangle \\ &= \langle \wedge, \langle A \rangle, \langle \vee, T(\neg P), T(Q) \rangle \rangle \\ &= \langle \wedge, \langle A \rangle, \langle \vee, \langle \neg, T(P) \rangle \rangle, \langle Q \rangle \rangle \\ &= \langle \wedge, \langle A \rangle, \langle \vee, \langle \neg, \langle P \rangle \rangle, \langle Q \rangle \rangle \end{aligned}$$

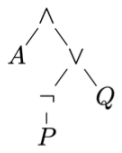

Fig. 3. Árbol de sintaxis de φ. A la derecha, representación gráfica.

B. Construcción.

La generación automática de una fórmula proposicional requiere de la conformación aleatoria de su estructura. Para construir una fórmula proposicional vamos a establecer dos parámetros:

1) Su *profundidad*. Es la cantidad máxima de niveles que tiene su árbol de sintaxis. Vamos a denotar este parámetro como n.
2) El *tamaño de su alfabeto*. Denotamos este parámetro como m. El alfabeto de una fórmula es el menor conjunto de variables que pueden aparecer en ella. Se denota $\Sigma = \{\phi_1, \phi_2, \dots, \phi_m\}$ y es tal que $supp(\varphi) \subseteq \Sigma$.

La profundidad de una fórmula determina un orden exponencial de complejidad en la cantidad de operadores que puede contener y por lo tanto de operadores que se tienen que resolver para evaluarla. La máxima cantidad de operadores en una fórmula proposicional con n niveles es $2^{n-1} - 1$. Por otro lado, el tamaño del alfabeto determina la cantidad de asignaciones que se tienen que resolver para evaluar la fórmula en todo su dominio, que para el caso de una fórmula proposicional es 2^m. El alfabeto de una fórmula se extrae de manera aleatoria del conjunto $\mathcal{P}$. Para la generación de una fórmula vamos a definir las siguientes funciones de selección aleatoria:

- $rand(\Sigma)$. Selecciona aleatoriamente alguna variable del alfabeto.
- $rand(op)$. Selecciona aleatoriamente algún elemento del conjunto de operadores $op = \{or, and, not, var\}$.

Para la construcción del árbol vamos a proceder de manera inductiva desde la raíz con la función de construcción *node* y el argumento n. En cada paso se hace la selección aleatoria de alguna operación (or, and o not) o el indicador de variable (var) para construir un nodo terminal, todos con la misma probabilidad de ser elegidos. También se actualiza la altura del árbol que se está construyendo de tal manera que cuando se alcance su último nivel (cuando $n = 1$) solo puedan ser generados nodos terminales:

$$node(n) = \begin{cases} \langle rand(\Sigma) \rangle & n = 1 \\ \langle rand(\Sigma) \rangle & n > 1, rand(op) = var \\ \langle \neg, node(n-1) \rangle & n > 1, rand(op) = not \\ \langle \wedge, node(n-1), node(n-1) \rangle & n > 1, rand(op) = and \\ \langle \vee, node(n-1), node(n-1) \rangle & n > 1, rand(op) = or \end{cases}$$

Así, $T \leftarrow node(n)$ construye aleatoriamente el árbol de sintáxis T con una altura máxima de n niveles.

C. Visualización.

Una fórmula es guardada y entendida por el árbol a través de su estructura, es decir su árbol de sintaxis. Sin embargo, se requiere que sea mostrada al usuario para que pueda ser leída e interpretada. Para ello usamos la función $string(T)$ que transforma el árbol T en una cadena. El operador $\cdot$ se usa para indicar concatenación.

$$string(T) = \begin{cases} \texttt{(} \cdot string(L) \cdot \texttt{or} \cdot string(R) \cdot \texttt{)} & T = \langle \vee, L, R \rangle \\ \texttt{(} \cdot string(L) \cdot \texttt{and} \cdot string(R) \cdot \texttt{)} & T = \langle \wedge, L, R \rangle \\ \texttt{not} \cdot string(S) & T = \langle \neg, S \rangle \\ ascii(\phi) & T = \langle \phi \rangle, \phi \in \Sigma \end{cases}$$

Esta cadena puede ser mostrada en pantalla para manifestar la expresión φ. En nuestra aplicación, esta cadena se genera en el formato LaTeX para renderizarla y posteriormente mostrarla con un formato profesional al usuario.

D. Evaluación.

En un caso de uso típico de nuestra herramienta, se pide al usuario que calcule algún resultado a partir de una fórmula que le es proporcionada. Para poder evaluar su respuesta es necesario que la aplicación haga primero una evaluación de esta misma. Se muestra una manera de evaluar fórmulas proposicionales.

Una fórmula proposicional se interpreta con respecto de alguna asignación de valores $\nu: \Sigma \to \mathbb{B}$, llamada una *valuación*. Si φ es una fórmula proposicional usamos la función $eval(\varphi)$ para evaluarla sobre su árbol de sintaxis:

$$eval(T) = \begin{cases} eval(L) \vee eval(R) & T = \langle \vee, L, R \rangle \\ eval(L) \wedge eval(R) & T = \langle \wedge, L, R \rangle \\ \neg eval(S) & T = \langle \neg, S \rangle \\ \nu(\phi) & T = \langle \phi \rangle, \phi \in \Sigma \end{cases}$$

La interpretación de las operaciones proposicionales se hace de acuerdo con las *tablas de evaluación* que se muestran en la fig. 4.

		$\vee$	$\wedge$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

	$\neg$
0	1
1	0

Fig. 4. Tablas de evaluación proposicional.

Vamos a generar todo el conjunto de valuaciones para evaluar φ en cada una de ellas con el procedimiento $allval$ de la fig. 5. Se asume el alfabeto $\Sigma = \{\phi_1, \phi_2, \dots, \phi_m\}$.

procedure *allval*(T,i)

1. **if** $i > m$ **then** *output*(ν, *eval*(T))
2. **else**
3. $\nu(\phi_i) \leftarrow 0$
4. *allval*($T, i+1$)
5. $\nu(\phi_i) \leftarrow 1$
6. *allval*($T, i+1$)
7. **end if**

Fig. 5. Generación de valuaciones.

IV. Integración de generador y solucionador de fórmulas proposicionales

Una característica esencial de este proyecto es que los problemas matemáticos son generados de manera automática cada vez que se solicitan. Es decir que no se cuenta con un banco de problemas, y cada vez que un problema es presentado al usuario, este es generado automáticamente. Esto hace que cada problema que sea virtualmente nuevo, y cuando se asigna una tarea, el conjunto de ejercicios sea diferente para cada usuario. Los algoritmos que generan planteamientos matemáticos pueden configurarse para determinar la complejidad del problema. En la fig. 6, muestra cómo se usó la formalización de la sección III para demostrar la funcionalidad y potencial de la plataforma, se generaron ejercicios donde el profesor o alumno configura la complejidad con la que es generada una formula, eligiendo el número de símbolos proposicionales n y la profundidad del árbol m.

Fig. 6. Interfaz de configuración de la complejidad del problema a generar.

En este momento es donde entra en ejecución el generador de fórmulas. Esta función llama al ejecutable del generador del ejercicio por medio de un subproceso de Python. Este ejecutable cual recibe tres argumentos principales: la ruta donde está almacenado el archivo con el generador, y los parámetros necesarios para su funcionamiento: n y m. Como respuesta, la aplicación genera una cadena que contiene tres tipos de información separada por el carácter ",": las variables utilizadas. La fórmula en formato MathLive y la respuesta simplificada de ceros y unos que determina la tabla de verdad de la fórmula proposicional generada. La plataforma despliega esta fórmula (fig. 7) y muestra la interfaz para que el usuario introduzca la respuesta.

V. Conclusiones y trabajo futuro

El uso de una herramienta como esta permite mejorar el proceso de transferencia y generación de conocimientos, la adquisición de habilidades y la facilitación para la generación de tareas, brindando un apoyo a profesores y estudiantes que imparten cursos del área matemática computacional. El profesor podrá generar y evaluar tareas de complejidad variable y con contenido único para sus estudiantes de manera automática, permitiéndole concentrarse en los procesos de transferencia de conocimiento, elaboración de materiales didácticos y atención de sus cursos. Por otro lado, el estudiante contará con una fuente ilimitada de problemas matemáticos para generar habilidades técnicas de manera activa. La plataforma necesita una formalización de cada tipo de problema para los generadores y solucionadores. Generar interfaces que permitan el diseño de forma ágil es un problema que se desea abordar en la siguiente versión de la plataforma.

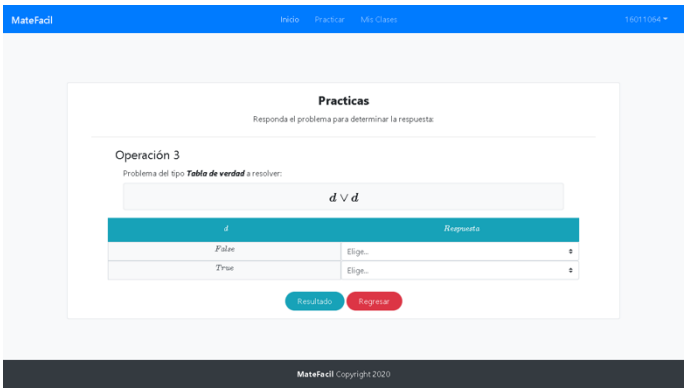

Fig. 7. Interfaz con ejercicio de práctica tabla de verdad.

VI. Agradecimientos

Agradecimientos al Instituto Tecnológico José Mario Molina Pasquel y Henríquez, Unidad académica Zapopan por el apoyo para la realización de este proyecto y en especial a los alumnos que trabajaron en la versión inicial de este proyecto: Luis M. Chávez Velasco, Jesús Castro Loyo, Jhonnatan J. Razo y Christian S. Mendoza.

VII. Referencias

[1] P. Harris Bonet, G. Romero Romero, M. A. Harros Bonet, y R. Llanos Díaz, "Análisis de las tendencias educativas con relación al desarrollo de las competencias digitales", *RiiTE Revista Interuniversitaria de Investigación en Tecnología Educativa*, No. 12, pp. 158–174, 2022, https://doi.org/10.6018/RIITE.520771

[2] A. Sánchez Rosal y G. Villamizar Acevedo, "Identificación de competencias educativas desarrolladas en contextos de aprendizaje virtual", *Revista Guatemalteca de Educación Superior*, vol. 5(1), pp. 13–22, 2021, https://doi.org/10.46954/revista-ges.v5i1.68

[3] M. Y. Torres Castro, P. Valera Yataco, M. I. Vásquez Valdivia, y G. S. Lescano López, "Desarrollo de las competencias matemáticas en entornos virtuales. Una Revisión Sistemática", *Alpha Centauri*, vol. 3(2), pp. 46–59, may 2022, https://doi.org/10.47422/ac.v3i2.80

[4] J. F. Romero Córdova y R. Arriazu Muñoz, "El aprendizaje de competencias en los MOOC. Una revisión sistemática de Literatura," *Revista Latinoamericana de Tecnología Educativa*, vol. 22(1), 2023. https://doi.org/10.17398/1695-288X.22.1.107