Instituto Politécnico Nacional

Centro de Investigación en Computación

Modelo de recomendación de problemas a realizar para competir en la Olimpiada de Informática

T E S I S

Que para obtener el grado de:

Maestría en Ciencias de la Computación.

P R E S E N T A :

Ing. Rodrigo Ruben Santiago Nieves

Directores de Tesis

Dr. Gilberto Lorenzo Martínez Luna

Dr. Adolfo Guzmán Arenas

SEPTIEMBRE 2015

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 29 del mes de julio de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Modelo de recomendación de problemas a realizar para competir en la Olimpiada de Informática"

Presentada por el alumno:

SANTIAGO	**NIEVES**	**RODRIGO RUBEN**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	1	3	0	1	2	9

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Gilberto Lorenzo Martínez Luna

Dr. Adolfo Guzmán Arenas

Dr. Juan Carlos Chimal Eguía

Dr. Salvador Godoy Calderón

Dr. Jesús Manuel Olivares Ceja

Dr. Rolando Menchaca Méndez

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

ESTADOS UNIDOS MEXICANOS
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día _24_ del mes de _Agosto_ del año_2015_, el (la) que suscribe _Rodrigo Ruben Santiago Nieves_ alumno(a) del Programa de _Maestría en Ciencias de la Computación_, con número de registro _B130129_, adscrito(a) al _**Centro de Investigación en Computación**_, manifiesto(a) que es el (la) autor(a) intelectual del presente trabajo de Tesis bajo la dirección del (de la, de los) **Dr. Gilberto Lorenzo Martínez Luna y Dr. Adolfo Guzmán Arenas** y cede los derechos del trabajo titulado _Modelo de recomendación de problemas a realizar para competir en la Olimpiada de Informática_, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del (de la) autor(a) y/o director(es) del trabajo. Este puede ser obtenido escribiendo a las siguientes direcciones _Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal, Col. Nueva Industrial Vallejo, Delegación Gustavo A. Madero, C.P. 07738, México D.F._ o al correo electrónico _rodrigonieves@live.com.mx_. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Santiago

Rodrigo Ruben Santiago Nieves

Nombre y firma del alumno(a)

Resumen

La Olimpiada Internacional de Informática (IOI) es una de las olimpiadas de la ciencia organizadas por la UNESCO. En México la Olimpiada Mexicana de Informática (OMI) es un concurso que busca a los mejores programadores mexicanos de nivel preparatoria que representaran a México en la IOI. Para poder entrenar se han creado páginas web con un conjunto de problemas, estas páginas tienen la capacidad de evaluar y entregar resultados de forma automática. Los problemas en estas competencias son de naturaleza algorítmica, y cuentan con su descripción, la especificación de la entrada y salida, para evaluar los problemas se crean un conjunto de entradas y se evalúa la salida del programa del alumno, los programas son ejecutados con limitantes en tiempo y memoria. Estas páginas constantemente incrementan sus problemas haciendo cada vez más difícil tener un orden de ellos, por lo tanto cada vez que un alumno entra a estas páginas termina confundiéndose y sin identificar que problemas le pueden ayudar a practicar nuevas habilidades.

El problema que pretende resolver la tesis es: Dado un conjunto de problemas P, un conjunto de usuarios U y una matriz UxP donde se indica para cada usuario $u \in U$ que ha enviado una solución al problema $p \in P$, la puntuación $r \in [0,\ 100]$. El problema es encontrar la mejor recomendación posible a cada usuario de aquellos problemas a resolver en un periodo de tiempo que le permitan mejorar su nivel de competencia.

Debido a las limitantes de tiempo se optó por realizar una simulación, la cual modela al usuario por su motivación y habilidades. La motivación del usuario indica cuantos problemas está dispuesto a intentar en cada ciclo, las habilidades indican en nivel del usuario en cada uno de los temas que se evalúan. La motivación puede ser modificada positivamente cuando el usuario resuelve un problema y puede decrecer cuando el usuario no puede resolver un problema. La habilidad puede incrementar cuando el usuario resuelve un problema.

En base a los registros históricos de la página desde enero del 2009 a enero del 2014 se calculan las probabilidades de resolver un problema dado el nivel del usuario y la probabilidad de que su nivel sea de n o superior dado que resolvió un problema. Con esas probabilidades la simulación determina cuando el usuario resuelve un problema que intenta y en caso de resolverlo cuando sube su nivel de habilidad.

Con la simulación se pueden probar diferentes modelos de recomendación, la simulación utiliza las funciones: inicia recomendador, realiza análisis y recomendación (idCompetidor), para interactuar con la simulación. Inicia recomendador se llama para que el recomendador inicialice todas sus variables, realiza análisis se llama al inicio de cada ciclo en el cual el recomendador puede hacer un análisis que tome mucho tiempo para acelerar el proceso de dar una recomendación, y recomendación recibe el identificador del competidor y tiene que regresar el identificador del problema recomendado.

Los modelos que se probaron con la tesis son:

1) Recomendaciones al azar: el recomendador elige un problema al azar para recomendar.
2) Recomendaciones en base a un experto: Se utiliza una lista de problemas ordenada por un experto en el área y dan las recomendaciones en ese orden.

3) Recomendaciones basadas en el número de inversiones en las historias de los usuarios: la historia de un usuario es la secuencia de los problemas que ha resuelto en el orden en el cual los resolvió. Una inversión es una pareja de problemas que aparece en la historia de dos usuarios distintos, que están en diferente orden en cada una de las historias. Contando el número de inversiones, cuantos problemas tienen en común y el complemento entre las historia, se elige para cada usuario cuales son los usuarios que conviene considerarlos como similares. La recomendación toma los problemas de los usuarios similares busca cual es el más frecuente en las historias de los usuarios similares que no ha resuelto el usuario y esa es la recomendación.
4) Recomendaciones basadas en la similitud de usuarios: se calcula para cada par de usuarios la similitud y basado en la premisa de que usuarios similares tendrán resultados similares se utilizan a los usuarios similares para predecir la calificación de los problemas. La recomendación es el problema que obtenga la mayor calificación en la predicción.
5) Recomendaciones basadas en la similitud de problemas: se calculan para cada par de problemas su similitud y basado en la premisa que un usuario obtendrá calificaciones similares en problemas similares, se utilizan los problemas que el usuario ha resuelto y la similitud entre problemas para predecir la calificación de estos. Se recomienda el problema con mayor puntuación predicha.
6) Recomendaciones basadas en la factorización de matrices: lo que se busca con este método es dividir la matriz de usuarios x problemas en el producto de tres matrices (usuarios x características, identidad características x características y características x problemas), se busca obtener las dos matrices que representan a los usuarios y a los problemas reduciendo el error cuadrático medio de la predicción por el producto de las matrices y la calificaciones que se ha obtenido. Se recomienda el problema con la mayor puntuación predicha.

Se simularon los 6 recomendadores con 100 usuarios y durante 30 ciclos (cada ciclo es el estimado de una semana). Junto a esto la simulación registra algunas variables las más importantes son:

- Recomendaciones: Número de recomendaciones que se dan durante toda la simulación.
- Éxitos: El número de problemas resueltos exitosamente.
- Fallos: El número de problemas fallados.
- Incrementos: El número de veces que los usuarios incrementaron su habilidad en algunos de los temas.
- Completos: El número de usuarios que completaron todos los problemas.
- Rendidos: El número de usuarios que su motivación llego a ser menor que 1 por lo tanto ya no continuaron en la simulación.
- ColdStart: El número de veces que el recomendador no tuvo suficiente información y fue necesario llamar a un segundo recomendador (para las pruebas se trataba de recomendaciones al azar).

Palabras clave: Sistema de recomendación, Olimpiada de Informática, Jueces en línea, filtros colaborativos.

Clasificación ACM (CCS 12): Information systems -> Information retrieval -> Retrieval task and goals -> Recommender systems

Abstract

The International Olympiad in Informatics (IOI) is one of the five international Olympiads patronized by the UNESCO. For Mexico the Mexican Olympiad in Informatics (OMI) is a contest which select the best Mexican programmers from high school who are going to represent Mexico at the IOI. In order to prepare students for these contest web pages (Online-judges) had been developed which have a set of problems, theses web pages are able to assess problems and deliver results automatically. Problems in these contests are oriented to algorithms, they are composed of their description of the problem, input and output. Assessing problems is done using a set of test cases created previously, codes are executed with limits in memory and execution time, finally the output is evaluated to asses that follow the output description. Online judges constantly add new problems which has resulted in an increase in disorder on those problems creating confusion for students and disorientation on how to progress.

The objective problem of this thesis is: Given a set of problems P, a set of users U and the matrix UxP where for each user $u \in U$ who has sent a solution to the problem $p \in P$ there is a rating $r \in [0, 100]$. Find the best recommendation possible for each users that in general increase the average motivation of each user and the average ability of users in a limited time.

Because of limits in time and ethics we decided to create a simulation instead of test with real students to evaluate different recommender systems, these simulation model user through his motivation and abilities. User's motivation address how many problems the user is going to try in each cycle. Abilities identify the level of the user in each topic to evaluate. Motivation depends on the problems solved or not solved in previous cycle, so it increase when the problems were solved correctly and decrease in the other case.

The simulation take values of Karelotitlan online-judge, a Mexican web page used for training, with its historical registers from January of 2009 until January 2014. We use the data to calculate the probabilities to solve a problem given the level of the user and the probability of having a level superior of the previous given that a problem p has been solved with that the simulation determines when a user solve a problem and given that solved a problem if he increases him level of ability.

With the simulation we could test different models of recommendation, each recommender use the functions: "start recommender", "do analysis" and "recommend (idUser)". Start recommender is called to initialize all variables, do analysis is called at the beginning of each cycle so the recommender could make an offline analysis, and recommend give the recommendation for each user.

We evaluate the following models:

1) Random recommendation: the recommender select a problem randomly to recommend.
2) Expert based recommendation: We use a list of problems created by an expert, and problems are recommended in that order.
3) Inversion based recommendation: We use user's history that is the sequence in which the user solved problems. Based on the number of inversion between history of users, the number of problems in common and the complement of problems we decided which are

the user that are more suitable to be considered as similar. The recommendation is given polling all the similar user.

4) User based recommendation: We calculate for each pair of users the similarity between them and based on the assumption of similar users will have similar scores we use the similar user scores to predict the score of the users.
5) Problem based recommendation: We calculate for each pair of problems the similarity between them and based on the assumption of the user will be have same scores in similar problems we use the scores of previous problems to predict the score of each problem.
6) Singular Value Decomposition based recommendation: We calculate an approximation to a factorization of scores' matrix, so we can use it to predict the score of each user at each problem.

We simulate all the recommendations with 100 users for 30 cycles (each cycle is an estimated of one week). And register different variables such as: recommendations, successes, failure, increments, completed, dropouts, cold starts.

Key words: Recommender system, Olympiad in informatics, Online-judge, collaborative filter.

ACM Classification (CCS 12): Information systems -> Information retrieval -> Retrieval task and goals -> Recommender systems.

Agradecimientos

A mis directores de tesis: Dr. Gilberto Martínez y Dr. Adolfo Guzmán, por su apoyo y orientación a lo largo de esta etapa.

A los miembros del jurado: Dr. Salvador, Dr. Jesús Olivares, Dr. Rolando Menchaca y Dr. Juan Carlos Chimal, por sus comentarios y observaciones que me permitieron mejorar mi trabajo y formación.

A mis amigos y compañeros que a lo largo de todo este tiempo mostraron su apoyo y disponibilidad, por levantarme el ánimo en los momentos de tristeza, por celebrar a mi lado en los momentos de alegría, por las observaciones a mi trabajo y sobre todo por su compañía.

A mi familia por siempre estar a mi lado.

A la Olimpiada Mexicana de Informática por inculcarme la curiosidad por las ciencias de la computación y permitirme ayudar a nuevas generaciones.

Al Centro de Investigación en Computación por todo su apoyo y atención administrativa y docente.

Al Instituto Politécnico Nacional por ser mi alma máter en la preparatoria, universidad y posgrado.

Al Consejo Nacional de Ciencia y Tecnología por el apoyo brindado, sin el cual estudiar un posgrado no hubiera sido posible.

Índice

Capítulo 1: Introducción

El hipocampo ayuda a guardar nuevos recuerdos en la corteza cerebral en un proceso llamado consolidación [1]. Los recuerdos son partes vivas del cerebro que respiran y están cambiando todo el tiempo. Siempre que se recuerda un recuerdo, éste cambia en un proceso llamado reconsolidación [2]. El proceso de consolidación toma la parte del cerebro en estado activo y lo guarda en la memoria a largo plazo, modificando las sinapsis en las dendritas de las neuronas. Estos recuerdos a largo plazo pueden quedar latentes por un largo periodo hasta que el recuerdo se recupera y se reintegra a la memoria a corto plazo mediante el proceso de reconsolidación. El recuerdo reintegrado está en un nuevo contexto el cual puede ser almacenado en la memoria a largo plazo alterando el viejo recuerdo. Los procesos de consolidación y reconsolidación ocurren también mientras dormimos. Por esto, es más efectivo espaciar el aprendizaje, en lugar de aprender todo al mismo tiempo [3].

Una manera de entrenar para los concursos de programación es utilizando jueces en línea, los cuales son páginas web con la capacidad de publicar problemas y evaluarlos de forma automática, estas páginas sirven para reconsolidar los temas previamente vistos, el problema de estos sistemas es que los problemas no están ordenados para que sea fácil encontrar algunos que se puedan resolver y ayuden a profundizar en el conocimiento del tema. Esta tesis busca responder a la pregunta ¿Es posible emitir recomendaciones de problemas para motivar a los usuarios a constantemente estar resolviendo problemas y así elevar el nivel de sus habilidades?

Descripción

En los últimos años los concursos de programación han fomentado el interés por la ciencia de la computación, generando talento de alta competitividad. En el caso de Mexico, la Olimpiada Mexicana de Informática ha juntado esfuerzos de diferentes personas alrededor del país con el objetivo de fomentar el interés por las ciencias de la computación.

Parte de los esfuerzos han dado como resultados diversas páginas web donde los alumnos tienen la posibilidad de practicar con problemas de naturaleza algorítmica y recibir retroalimentación. Estos sistemas constantemente incrementan el número de problemas lo cual hace cada vez más difícil poder identificar qué problema resolver para tener el mejor progreso.

La presente propuesta es para desarrollar un modelo de recomendación para los usuarios de sistemas de entrenamiento para participar en competencias de programación basado en el historial de problemas resueltos y no resueltos.

Contexto a los concursos de programación

Olimpiada Internacional de Informática (IOI)

La idea de iniciar las olimpiadas internacionales de Informática para estudiantes fue propuesta en la vigésimo cuarta conferencia general de la “Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura” (UNESCO, por sus siglas en inglés) en París por el Profesor Sendov, delegado de Bulgaria en Octubre de 1987. Este plan fue incluido en el quito programa Principal de la UNESCO para el bienio de 1988-1989 (sección 05 215). En mayo de 1989, la UNESCO inicio y patrocino la primera Olimpiada Internacional de Informática (IOI, por sus siglas en inglés), la cual fue realizada en Bulgaria [4].

La IOI es una de las cinco olimpiadas internacionales de la ciencia. La meta principal de la IOI es estimular el interés por la informática (ciencias de la computación) y las tecnologías de la información. Otra importante meta es reunir alumnos excepcionalmente talentosos desde varios países y compartirles experiencias científicas y culturales.

La IOI es organizada anualmente en y por uno de los países participantes. Cada país participante típicamente envía una delegación de cuatro concursantes y dos adultos acompañantes. Los estudiantes compiten individualmente y tratan de maximizar su puntuación resolviendo un conjunto de problemas informáticos durante dos días de competición. Eventos culturales y recreacionales son organizados en los días restantes.

Las tareas de la competencia son de naturaleza algorítmica: sin embargo, los concursantes tienen que mostrar habilidades básicas en tecnologías de la información como análisis de problemas, diseño de algoritmos y estructuras de datos. Los ganadores de la IOI pertenecen a los mejores jóvenes científicos computacionales en el mundo.

Las Regulaciones de la IOI (IOI Regulations) constituye la definición oficial de la IOI. El Comité Científico Internacional (ISC, por sus siglas en inglés) es responsable del contenido científico de la IOI. El comité internacional supervisa la IOI.

La siguiente tabla lista las sedes de la IOI.

Año	Ciudad sede	País sede	Países Participantes	Número de concursantes	Fechas
1989	Pravetz	Bulgaria	13	46	16 al 19 de Mayo
1990	Minsk	Bielorrusia	25	100	15 al 21 de Julio
1991	Athens	Grecia	26	69	19 al 25 de Mayo
1992	Bonn	Alemania	51	171	11 al 21 de Julio
1993	Mendoza	Argentina	43	155	16 al 25 de Octubre
1994	Haninge	Suecia	49	189	3 al 10 de Julio
1995	Eindhoven	Holanda	51	210	26 de Junio al 3 de Julio
1996	Veszprém	Hungría	57	220	25 de Julio al 2 de Agosto
1997	Cape Town	Sudáfrica	63	221	30 de Noviembre al 7 de Diciembre
1998	Setúbal	Portugal	68	241	5 al 12 de Septiembre
1999	Antalya-Belek	Turquía	65	253	9 al 16 de Octubre
2000	Beijing	China	72	278	23 al 30 de Septiembre
2001	Tampere	Finlandia	74	272	14 al 21 de Julio
2002	Yong-In	Corea del Sur	75	276	18 al 25 de Agosto
2003	Wisconsin	Estados Unidos	69	265	16 al 23 de Agosto
2004	Athens	Grecia	77	291	11 al 18 de Septiembre
2005	Nowy Sacz	Polonia	72	276	18 al 25 de Agosto

2006	Yucatán	México	75	282	13 al 20 de Agosto
2007	Zagreb	Croacia	75	285	15 al 22 de Agosto
2008	Cairo	Egipto	73	283	16 al 23 de Agosto
2009	Plovdiv	Bulgaria	76	301	8 al 15 de Agosto
2010	Waterloo	Canadá	80	297	14 al 21 de Agosto
2011	Pattaya	Tailandia	78	303	22 al 29 de Julio
2012	Sirmione	Italia	81	310	23 al 30 de Septiembre
2013	Brisbane	Australia	80	299	6 al 13 de Julio
2014	Taipei	Taiwán	81	311	13 al 20 de Julio

Tabla 1 Ciudades y países donde la IOI se ha realizado [5].

A continuación se muestran la lista de las próximas sedes de la IOI

Año	Ciudad sede	País sede
2015	Astana	Kazakstán
2016	Kazan	Rusia
2017	Tehran	Irán
2018	Tokio	Japón

Tabla 2 Ciudades y países designados como futuras sedes de la IOI [5]

Olimpiada Mexicana de Informática (OMI)

La Olimpiada Mexicana de Informática (OMI) es un concurso a nivel nacional para jóvenes con facilidad para resolver problemas prácticos mediante la lógica y el uso de computadoras, que busca promover el desarrollo tecnológico en México y encontrar a los mejores programadores, quienes formarán la selección mexicana para participar en las próximas Olimpiadas Internacionales de Informática(IOI) [6].

La OMI es un concurso en el que, sobre todo, se requiere tener facilidad, habilidad y voluntad de resolver problemas, utilizando la lógica, el ingenio y las computadoras.

Los objetivos que tiene la OMI son los siguientes:

- Fomentar entre los estudiantes de nivel medio superior del país, el interés por la informática y las ciencias de la computación.
- Promover el desarrollo tecnológico en México.
- Encontrar jóvenes talentosos dentro de estas áreas para darles apoyo en sus estudios y guía en sus inquietudes.
- Encontrar a la mejor selección para representar a México en las Olimpiadas Internacionales de Informática (IOI) que se llevan a cabo año tras año en alguno de los países participantes.
- Promover la amistad, convivencia e intercambio tecnológico entre jóvenes de todo el país con intereses comunes en la programación.

En la OMI pueden participar todos aquellos jóvenes que cumplan con:

- Estar inscritos en alguna escuela de nivel medio, escolarizado o no escolarizado (primaria, secundaria y preparatoria).
- Ser mexicanos

- Ser menor de 20 años a la fecha del concurso internacional (verano del siguiente año del que se inicia la OMI).
- Estar a lo más inscrito en segundo año de preparatoria o su equivalente.

El concurso consiste en dos días de exámenes, el primer día es un examen acerca de Karel mientras que el segundo día es un examen de naturaleza algorítmica que es resuelto en algún leguaje de programación autorizado por la Olimpiada Mexicana de Informática (los lenguajes que comúnmente se utilizan son: Pascal, C y C++). Para cada examen el competidor cuenta con 5 horas para poder enviar sus soluciones y para cada examen se crean 4 problemas.

La asignación de medallas se realiza en base a las mismas reglas de la Olimpiada Internacional. Procurando que un doceavo de los concursantes obtengan medalla de oro, dos doceavos medalla de plata y 3 doceavos medalla de bronce.

Karel

Karel el robot es un emulador de un robot, distribuido como software educativo de código abierto dirigido a principiantes en el estudio de lenguajes de programación, dando al iniciado sólidas bases en cuanto al diseño en un ambiente de programación estructurada como Pascal. Fue creado por Richard E. Pattis en su libro “Karel the Robot: A Gentle Introduction to the Art of Programming” Pattis usó el lenguaje en sus clases en la Universidad Carnegie Mellon, con el objetivo de que sus estudiantes aprendiesen a pensar de manera ordenada y eficiente [7].

Evaluación de Problemas

La competencia de la IOI toma lugar durante dos días los cuales son precedidos y sucedidos por un día de no competencia. [8, p. S6]

El país anfitrión debe dar a los concursantes la oportunidad de practicar en el equipo de la competencia o algún equipo similar previo a los días de competencia.

El comité científico tiene que presentar de forma impresa los procedimientos de la competencia y los procedimientos de evaluación de los problemas en la junta apropiada para su aprobación por la asamblea general. Los líderes de las delegaciones pueden traducir las versiones aprobadas de los procedimientos y problemas de la competencia a su lenguaje nativo sin dar información extra.

Los concursantes requieren resolver los problemas de la competencia, junto con cualquier traducción y materiales permitidos por los procedimientos de la competencia. No debe existir comunicación, información o cualquier otro que no sea permitido por los procedimientos de la competencia.

Los concursantes deben obedecer los procedimientos de la competencia.

Los líderes de las delegaciones deben de estar presentes al menos la primera mitad del examen en cada día de competencia desde el momento en que los problemas sean dados a conocer para poder traducir al inglés las preguntas de los concursantes, el comité científico es el encargado de responder estas preguntas o en caso de ser necesario serán respondidas por los miembros de asamblea general, con las respuestas Si, No o Sin comentarios.

La evaluación es realizada directamente después de terminada la competencia con los procedimientos de evaluación. Si el líder de la delegación no está de acuerdo con la evaluación de

los problemas, el desacuerdo debe ser presentado de acuerdo con los procedimientos de evaluación.

El comité organizador ejecuta la evaluación de los problemas de la competición de acuerdo con los procedimientos de evaluación.

La salida de la evaluación de cada elemento de los problemas es registrado en la forma de evaluación. La forma de evolución es distribuida a los concursantes mediantes los líderes de las delegaciones. La información de la evaluación y las soluciones de los concursantes se hace disponible para los líderes.

El número de puntos obtenido por el concursante no debe hacerse público antes de la ceremonia de premiación.

Antes de la ceremonia de premiación de la IOI el comité científico tiene que hacer una propuesta para la asamblea general en relación con la distribución de las medallas con base a la cantidad de puntos logrados por los concursantes. La asamblea general debe confirmar las puntuaciones. No más de la mitad de los concursantes recibirá medallas y la distribución se realizara de la siguiente manera:

- Alrededor de un doceavo de los concursantes recibe medalla de oro.
- Alrededor de un sexto de los concursantes recibe medalla de plata.
- Alrededor de un cuarto de los concursantes recibe medalla de bronce.

En los últimos años, para cada día de competencia, se seleccionan cuatro problemas cada uno con un valor de 100 puntos, lo cual da una puntuación máxima de 800 puntos en los dos días de competencia.

Para evaluar a los concursantes, previamente se generan un conjunto de entradas, se evalúa el código del concursante con cada una de las entradas, si el programa generado por el concursante da una salida considerada valida se le otorgan los puntos correspondientes a esa entrada. En ocasiones se agrupan entradas y solo se otorgan los puntos correspondientes si y solo si se obtiene una salida válida para cada entrada en el grupo.

Los programas tienen limitaciones de ejecución en tiempo y memoria, estos límites son especificados en los procedimientos de evaluación.

En algunos años se realizan variantes en la evaluación, por ejemplo: el sistema debe de interactuar con alguna librería, se le entregan las entradas al concursante y este debe entregar la salida, se identifica el resultado con mayor puntuación y a partir de esa puntuación se genera una fórmula que indica cuantos puntos les corresponde a cada concursante. Cualquiera que sea la forma de evaluar al final se entrega una puntuación entre 0 y 100 para cada problema.

Participación de México en la IOI

Mexico ha participado en forma ininterrumpida en las Olimpiadas Internacionales de Informática desde el año 1993, durante estos años se han ganado 1 medalla de oro, tres medallas de plata y trece medallas de bronce. Estos resultados son considerados bajos en comparación a la población estudiantil de México [9].

A continuación se muestran los primeros 7 lugares de las últimas 5 olimpiadas internacionales

Año	1er Lugar	2º Lugar	3er Lugar	4º Lugar	5º Lugar	6º Lugar	7º Lugar
2013	China	Rusia	Corea	USA	Bulgaria	Iran	Rumania
2012	China	Rusia	USA	Japón	Bielorrusia	Polonia	Rumania
2011	China	Rusia	USA	Taiwán	Croacia	Japón	Bielorrusia
2010	USA	China	Rusia	Japón	Polonia	Czech Rep.	Japón
2009	China	USA	Japón	Taiwán	Corea	Polonia	Rusia

Tabla 3 Lista de los primeros 7 lugares en la IOI en las últimas 5 Olimpiadas de Informática [9]

Como se puede observar claramente los países que se encuentran constantemente en los primeros lugares son China, Rusia y Estados Unidos. Los cuales han sido el modelo a seguir por México y han inspirado el sistema de selección mexicano.

La puntuación total en cada Olimpiada Internacional suele variar, por lo tanto la medida más apropiada para medir el avance de los diferentes países en las diferentes Olimpiadas Internacionales son los puntos sobre la media. A continuación se muestra una tabla con los puntos sobre la media obtenidos por México y el país campeón del respectivo año.

Año	Puntos(P) México	País campeón	Puntos(P) campeón	Media(M) individual	P/M MEXICO	P/M CAMPEON
1998	500	Slovakia	2700	310	1.61	8.71
1999	245	China	1539	135	1.81	11.40
2000	620	Rusia	2560	248	2.50	10.32
2001	272	Slovakia	1492	143	1.90	10.43
2002	348	Corea	1424	135	2.57	10.55
2003	344	Corea	1488	173	1.99	8.60
2004	670	China	2085	265	2.53	7.87
2005	872	China	2256	275	3.17	8.20
2006	653	China	1686	219	2.98	7.70
2007	686	China	2067	187	3.67	11.05
2008	528	China	1711	127	4.16	13.47
2009	1419	China	2716	399	3.56	6.8
2010	1868	USA	2821	537	3.48	5.25
2011	1145	China	2083	276	4.15	7.54
2012	629	China	1847	157	4.00	11.76

2013	1011	China	2196	220	4.59	9.98
2014	1194	China	2135	223	5.35	9.57

Tabla 4 Puntos sobre la media México y país campeón [10]

Para poder visualizar el avance de México a continuación se muestra una gráfica con los puntos sobre la media de México y las medallas ganadas.

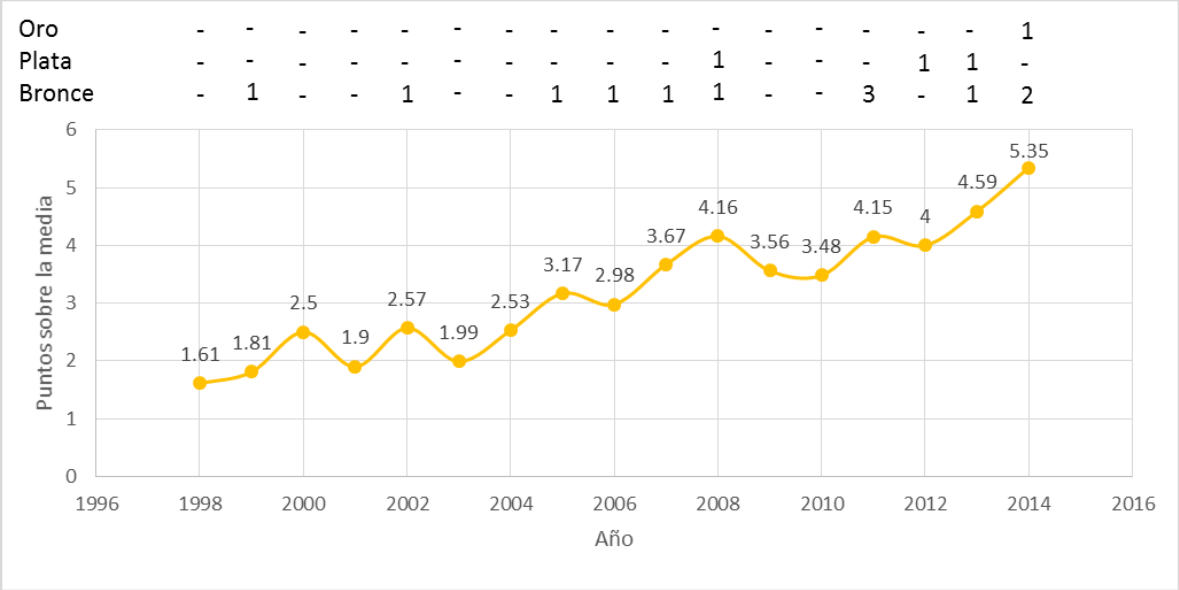


Ilustración 1 Gráfica Puntos sobre la media y medallas de Mexico en las olimpiadas de informática desde 1998 [10]

Otra forma de poder visualizar el avance de México en las Olimpiadas de Informática es mediante la posición relativa en cada año, a continuación se muestra la gráfica de las posiciones relativas que ha obtenido México en la Olimpiada de Informática.

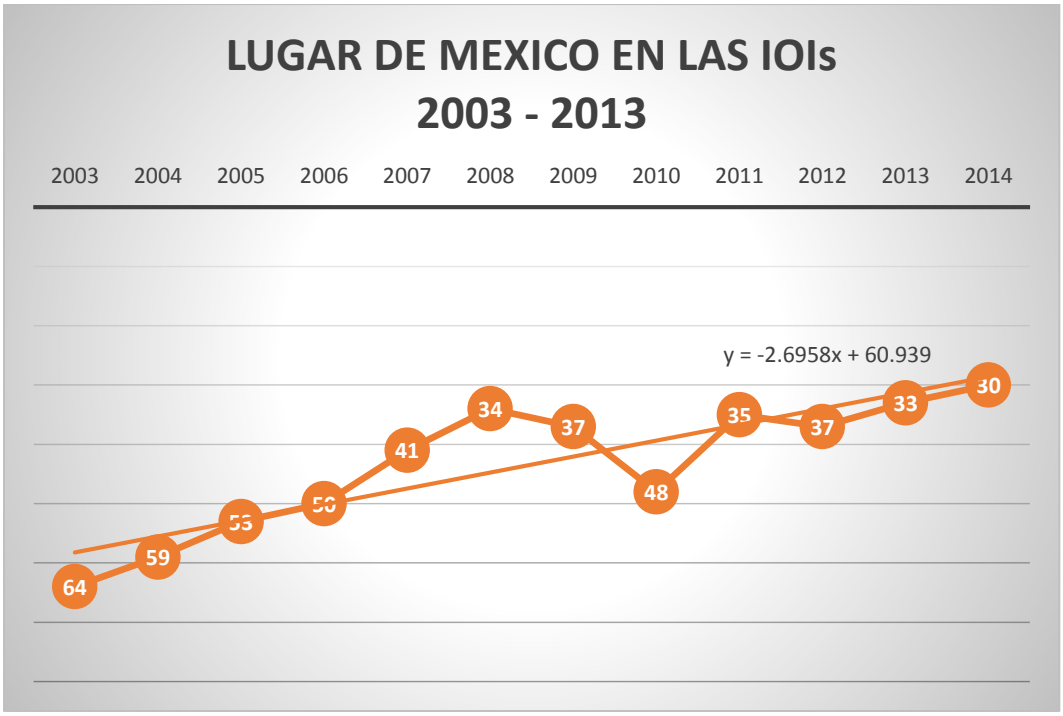


Ilustración 2 Gráfica de Lugar de México en las Olimpiadas Internacionales de Informática [10]

Proceso de selección de México

El proceso de selección de México es un proceso que toma alrededor de un año, las etapas son las siguientes:

- Inscripciones por internet.
- Aplicación de examen por internet que cubre los conceptos básicos de algebra, geometría y lógica.
- Selección estatal, cada estado de la república se encarga de capacitar y evaluar a los estudiantes de su estado. Seleccionando a 4 alumnos que participarán en la Olimpiada Mexicana de Informática.
- Olimpiada Mexicana de Informática
- Después de este concurso se crea la preselección nacional, la cual está constituida por todos los alumnos que obtuvieron medalla de oro o plata y los alumnos que ganaron medalla de bronce y puedan participar en la siguiente Olimpiada Mexicana de Informática.
- El proceso preselectivo consiste en una serie de entrenamientos y exámenes de los cuales se eligen a los alumnos hasta quedar 4 alumnos que serán parte de la selección nacional para participar en la IOI.

En los últimos años se ha buscado otorgar la mismas oportunidades a todos los alumnos del país, la principal diferencia entre cada estado es la capacidad para entrenar a los estudiantes. Por eso se ha iniciado con entrenamientos anuales para profesores, que tiene como objetivo capacitar a los profesores de todos los estados del país, de esta manera podrán capacitar a sus alumnos en los temas que son requeridos por la Olimpiada Internacional de Informática.

Sistemas de entrenamiento para competencias de programación

En los últimos años han surgido diversos sistemas de entrenamiento para competencias de programación, estos sistemas se caracterizan por tener diversos problemas en los cuales se puede subir el código fuente de una solución y esta es evaluada automáticamente. Estos sistemas normalmente organizan concursos de forma periódica, así se sigue incrementando los problemas que soporta el sistema de forma periódica.

Estos sistemas están enfocados principalmente en la Olimpiada internacional de Informática y en el ACM International Collegiate Programming Contest (ICPC).

Dentro de estos sistemas se destacan:

- Sphere Online judge (SPOJ [11]): SPOJ soporta más de 45 Lenguajes de programación: entre ellos C, C++, Pascal, JAVA, C#, Perl, Python, Ruby, Haskell, Acaml y otros más, un conjunto de 13,000 problemas.
- CODECHEF [12]: Sitio web que busca ser una comunidad global, es anfitriona de concursos, entrenamientos y eventos de programadores alrededor del mundo.
- CodeForces [13]: es un proyecto que busca unir a personas interesadas en participar en concursos de programación, codeforces es una red social dedicada a programación y a los concursos de programación, por el otro lado, es una plataforma que puede ser usada por los usuarios para prepararse.
- TopCoder [14]: TopCoder es una comunidad que reúne a expertos alrededor del mundo en diseño, desarrollo y ciencias de datos para trabajar en problemas interesantes y desafiantes por diversión y premios. TopCoder busca ayudar a sus usuarios a mejorar sus habilidades, demostrar sus conocimientos y ganar recompensas.
- USACO training [15]: Esta página otorga una serie de problemas con el objetivo de entrenar a los participantes de estados unidos para la Olimpiada Internacional de Informática.

En México se han desarrollado diversos sistemas para el entrenamiento en competencias de programación de las cuales destacan:

- Karelotitlán [16]: Es una página diseñada para que alumnos que participan en la olimpiada mexicana de informática puedan practicar las habilidades de programación con el simulador de robot Karel, el cual es programado en lenguajes similares a Pascal o Java, por medio de problemas que abarcan diferentes temas, el sistema tiene la capacidad de evaluar de forma automática los problemas por medio de casos preestablecidos para cada uno de los problemas. Este sistema está especializado en evaluar problemas de Karel, esta página cuenta con alrededor de 125 problemas y 15,000 usuarios.
- OmegaUp [17]: Es un proyecto web enfocado a elevar el nivel de competitividad de desarrolladores de software en América Latina mediante la resolución de problemas de algoritmos, con un enfoque competitivo y divertido a la vez. Ha sido utilizado para poder evaluar la Olimpiada Mexicana de Informática los últimos 3 años, cuenta con alrededor de 700 problemas y 1,700 usuarios.

Problema

El problema a resolver en esta tesis es el siguiente:

Dado un conjunto de problemas P, un conjunto de usuarios U y una matriz UxP donde se indica para cada usuario $u \in U$ que ha enviado una solución al problema $p \in P$, la puntuación $r \in [0, 100]$.

El problema es encontrar la mejor recomendación posible a cada usuario de aquellos problemas a resolver en un periodo de tiempo que le permitan mejorar su nivel de competencia.

Justificación

Los sistemas de entrenamiento han incrementado la cantidad de problemas en forma constante volviéndose cada vez más complicado tener un orden en el que se les presentan al estudiante.

Esto lleva a la necesidad de una guía para poder elegir cual es la mejor secuencia de problemas que puede ayudar en el proceso del entrenamiento.

Objetivos

Objetivo general

Desarrollar un prototipo capaz de comunicarse con alguno de los sistemas de entrenamiento para obtener la información de los usuarios para así analizar los datos de los problemas y poder emitir una secuencia de problemas recomendada.

Para lo cual se evalúan diversos modelos de recomendación para identificar cuales motivan a los estudiantes a resolverlos (incrementan el número de problemas intentados durante un período de tiempo) y mejoran sus habilidades (incrementan la dificultad de los problemas que pueden resolver).

Objetivos particulares

- Utilizar una herramienta de extracción, transformación y carga (ETL por sus siglas en inglés) que se comunique con un sistema de entrenamiento.
- Generar un módulo encargado de simular el proceso de entrenamiento de varios usuarios.
- Generar un módulo de visualización de parámetros generadas por la simulación.
- Generar y programar 5 modelos de recomendación.
- Comparación de recomendaciones generadas por los diferentes modelos de recomendación.

Alcances

La trascendencia de esta investigación es poder brindar una herramienta, que pueda ser utilizada por diversos estudiantes en todo el país, para poder guiar un proceso de autoentrenamiento y así preparar a alumnos que no cuenten con el apoyo de una persona especializada.

Esta herramienta también ayudará a la capacitación de nuevas generaciones en el área de las ciencias de la computación, fomentando el interés por la informática.

Capítulo 2: Estado del arte en la recomendación de problemas

Modelos de recomendación en otras áreas

En 2005 Adomacivius [18] definió a los sistemas de recomendación de la siguiente manera:

Dado $\boldsymbol{C}$ el conjunto de todos los usuarios, $\boldsymbol{S}$ el conjunto de todos los posibles elementos que pueden ser recomendados y $\boldsymbol{u}$ la función de utilidad que mide la utilidad del elemento $\boldsymbol{s}$ para el usuario $\boldsymbol{c}$.

u:$CxS \rightarrow R$ donde R es un conjunto ordenado.

Entonces para cada usuario $c \in C$ elegir un elemento $s' \in S$ que maximice la utilidad del usuario, formalmente:

$$\forall c \in C,\ s' = argmax_{s \in S} u(c, s)$$

Con esta definición la diferencia entre los problemas de recomendación se centran principalmente en la función de utilidad y en el contexto del problema.

Historia de los sistemas de recomendación

En el prólogo del libro "Recommender Systems" [19] Joseph A. Konstan hace una recapitulación de lo que él considera han sido las cuatro fases de los sistemas de recomendación, las cuales son:

1) Exploración orientada en sistemas
2) Rápida comercialización (los retos de escala y valores)
3) Explosión de la investigación (los sistemas de recomendación se vuelven comunes)
4) Avanzando (recomendadores en contexto)

La idea de recolectar las opiniones de millones de personas en línea para ayudar a encontrar contenido más útil e interesante surgió a principios de los noventas. En varias áreas y diferentes formas. El sistema PARC Tapestry [20] en 1992 introdujo la idea de filtros colaborativos y mostro como anotaciones explicitas y datos de comportamiento implícito pueden ser obtenidos y utilizados en una base de datos para generar filtros personales. Dos años después el sistema de GroupLens [21] mostraba un acercamiento de filtros colaborativos que pueden ser distribuidos en una red y a su vez automático sobre un servicio de noticias (Usenet). Por su lado en 1995 el sistema Ringo [22] en el Massachusetts Institute of Technology (MIT) hicieron lo mismo que GoupLens pero para álbumes de música y artistas, estos sistemas utilizaban técnicas de automatización similares. El algoritmo de los k vecinos más cercanos identifica usuarios con gustos similares y combinan sus calificaciones en promedios cargados personalizados, esto probó ser muy eficiente por lo cual se convirtió en el algoritmo estándar para comparar filtros colaborativos.

La primera fase, exploración orientada en sistemas, fue caracterizada por la demostrada eficacia de los sistemas de recomendación generando gran emoción en el avance del campo, sistemas como FindMe [23] en 1996 que no solo fue basado en filtros colaborativos sino también emplearon conocimiento basado en el sistema. Uno de los eventos importantes que se realizaron en esta fase fue "the Collaborative Filtering Workshop" en Berkeley en marzo de 1996, en la cual se identificó un nuevo problema que rápidamente fue conocido como sistemas de recomendación.

La siguiente fase, rápida comercialización, los sistemas de recomendación surgieron durante un periodo de rápida expansión del internet surgiendo empresas como "Agents, Inc." fundada por el

grupo Pattie Maes en el MIT y Net Perceptions fundada por el grupo GroupLens en Minnesota, estas nueva compañías empezaron a enfrentar problemas del mundo real y fue necesario mostrar predicciones adecuadas y no afectar la eficiencia de las páginas de esa era, estos sistemas tenían que manejar millones de usuarios y productos y cientos o miles de transacciones por segundo. Diversos algoritmos surgieron durante esta era como algoritmos de correlación basados en elementos y enfoques de reducción de dimensiones, también la investigación se enfocó en evaluar recomendadores basados en listas de las mejores n recomendaciones. La investigación se enfrentó a problemas relacionados con calificaciones implícitas, nuevos usuarios, nuevos elementos, experiencia de usuarios, confianza, explicaciones y transparencia.

La tercera fase, explosión de la investigación, fue aproximadamente entre el año 2000 y 2005, muchas compañías dedicadas a sistemas de recomendación cerraron debido a la explosión de la burbuja del internet, unas cuantas que lograron generar herramientas que abarcaban diversas áreas pudieron permanecer con un gran uso en comercio electrónico, ventas minoristas y conocimiento de administración. Al mismo tiempo la investigación en sistemas de recomendación se amplió con la inclusión de diversos enfoques de diversas disciplinas como inteligencia artificial, recuperación de la información, minería de datos, seguridad, privacidad, negocios y mercadotecnia. Parte de las investigaciones fue motivada por el premio de Netflix que consiste en 1 millón de dólares por un incremento del 10% en la precisión de la predicción.

La fase actual, Avanzando, surgió la polémica acerca del premio de Netflix ya que algunos investigadores argumentaban que la evaluación de la precisión de la predicción planteada por Netflix no era la mejor posible. En 2006 MyStrands organizo Recommenders06 que es una escuela de verano sobre el presente y futuro de los sistemas de recomendación. En 2007 fue organizada la primera conferencia de sistemas de recomendación (ACM Recommender Systems Conference), en estas conferencias ha existido interés en ver las recomendaciones en su contexto, construcción de nuevas herramientas de investigación para fundamentar los recomendadores en un entendimiento de como las personas interactúan en las organizaciones y negocios. Uno de los artículos más famosos de estas conferencias es “Collaborative Prediction and Ranking with Non-random Missing Data” [24] en el cual se plantea un método para evaluar algoritmos de recomendación y el articulo más citado es “Evaluating collaborative filtering recommender systems” [25] que explica como empatar la evaluación con las necesidades del usuario.

Introducción a sistemas de recomendación

Un sistema de recomendación es un software que determina que elemento mostrar a cada usuario del sistema. Por esta razón, es necesario que este sistema cuente con un modelo de usuario, también conocido como perfil, que es obtenido mediante preferencias explicitas o implícitas del usuario. Estos sistemas tienen la capacidad de otorgar recomendaciones personalizadas, es decir, cada usuario recibe una recomendación en base a su perfil. El enfoque que predomina en la actualidad es el basado en la comunidad o colaborativo, que busca apoyarse en la información que se obtiene de una gran comunidad de usuarios con diferentes opiniones, gustos y comportamiento.

Recomendaciones colaborativas

Estos sistemas están basados en la idea de que si los usuarios compartieron un gusto en el pasado entonces ellos tendrán gustos similares en el futuro. Por ejemplo si tenemos dos personas A y B, las compras que han realizado estos dos usuarios son muy similares si A ha comprado un libro que B no

tiene por lo tanto es lógico que ese libro le interese a B, el conjunto de libros que potencialmente B estará interesado en comprar forma una lista de recomendación. Como se puede observar estos dos usuarios están colaborando de manera implícita y esta técnica es llamada filtros colaborativos.

Algunas de las preguntas que surgen con este tipo de sistemas son:

- ¿Cómo encontrar usuarios con gustos similares a los usuarios que requieren recomendaciones?
- ¿Cómo se mide la similitud entre usuarios?
- ¿Qué se tiene que haces con los usuarios nuevos?
- ¿Cómo lidiar con nuevos elementos o que nadie ha comprado?
- ¿Qué pasa si tenemos solo pocas calificaciones para un producto?
- ¿Qué otras técnicas, aparte de buscar usuarios similares, pueden ser usadas para realizar predicciones acerca de que elementos el usuario preferirá?

Una de las ventajas de estos sistemas es que no es necesario tener ningún conocimiento acerca del contenido de los elementos, por ejemplo en un recomendador de libros utilizando este enfoque no necesita saber de qué trata el libro, su género o su autor. Así mismo su desventaja es que conocer este tipo de información puede ser útil para dar una recomendación más efectiva.

Recomendaciones basadas en contenido

En general los sistemas de recomendación pueden tener 2 objetivos diferentes, pueden ser utilizados para estimular el interés de los usuarios a comprar algún producto o pueden ser utilizados para lidiar con la sobre carga de información, ya que su objetivo es seleccionar los elementos más interesantes de una larga lista de elementos. Por lo tanto una forma de atacar este problema es darle una calificación a los elementos de acuerdo a su contenido. Las recomendaciones basadas en contenido, buscan obtener descripciones de los elementos y un perfil que asigna la importancia a estas características.

Las preguntas a responder en estos sistemas de recomendación son:

- ¿Cómo el sistema puede adquirir y constantemente mejorar los perfiles de los usuarios?
- ¿Cómo empatar los elementos con los intereses de los usuarios?
- ¿Qué técnicas se pueden usar para poder extraer o aprender las características de los eleméntenos y evitar la captura manual de ellos?

En comparación con las recomendaciones colaborativas, estas recomendaciones tiene las ventajas de no se requiere una gran cantidad de usuarios para poder dar buenas recomendaciones y los nuevos elementos pueden ser recomendados inmediatamente. Debido a que algunas de las características de los elementos pueden ser muy difíciles de obtener estos sistemas de recomendación pueden terminar siendo muy difíciles de mantener y muy costosos.

Recomendaciones basadas en conocimiento

En ocasiones los sistemas son utilizados una sola vez, por lo tanto la posibilidad de generar un perfil de usuario es muy complicada, por ejemplo, en la venta de electrónicos el usuario solo accede a la página para comprar un tipo de producto, si el sistema no tiene información acerca del usuario la recomendaciones que puede dar son la de los productos más vendidos, he de ahí que estos sistemas

buscan obtener información del usuario mediante cuestionarios para identificar las necesidades del usuario y así poder otorgar una recomendación. Estos sistemas tienen una carga mayor con la interacción humano-máquina para poder identificar las necesidades del usuario.

Las preguntas que comúnmente surgen al desarrollar este tipo de sistemas son:

- ¿Qué tipo de conocimiento del área puede ser representada?
- ¿Qué mecanismos pueden ser usados para selección y calificar elementos basados en las características del usuario?
- ¿Cómo se obtendrá el perfil del usuario en el área en el cual no se tiene ningún historial de compras del usuario y como utilizar las preferencias explicitas del usuario en cuenta?
- ¿Qué patrones de interacción pueden ser usados para interactuar con el sistema de recomendación?
- ¿De qué manera podemos personalizar el dialogo con el usuario con el objetivo de mejorar el proceso de obtención de las preferencias del usuario?

Enfoque hibrido de recomendaciones

Este enfoque pretende combinar al enfoque colaborativo y el basado en contenido para otorgar una recomendación más precisa. Este enfoque busca responder a las siguientes preguntas:

- ¿Qué técnicas pueden ser combinadas y cuáles son los prerrequisitos para poderlas combinar?
- ¿Existen otros diseños híbridos?
- ¿los resultados de diferentes técnicas pueden ser ponderadas y se puede calcular esta ponderación dinámicamente?

Evaluación de sistemas de recomendación

Todos los sistemas de recomendación tienen el objetivo de mejorar las recomendaciones que se realizan, por tal motivo una de las preguntas más importantes a responder es ¿Cómo evaluar la calidad de las recomendaciones?, por eso es importante poder responder a las preguntas:

- ¿Qué diseños de investigación se aplican para la evaluación de sistemas de recomendación?
- ¿Cómo los sistemas de recomendación pueden ser evaluados usando experimentos basados en datos históricos?
- ¿Qué métricas son aplicables para la evaluación de diferentes metas?
- ¿Cuáles son las limitaciones de las técnicas de evaluación actuales?

Filtros Automáticos Colaborativo (Automated Collaborative Filterins ACF)

Los sistemas de filtrado automáticos colaborativos (ACG, por sus siglas en inglés) predicen la afinidad de una persona para elementos o información conectando los intereses que una persona tiene con los intereses de la comunidad de personas que comparten sus preferencias. En el artículo “Explaining Collaborative Filtering Recommendations” [26] se hace referencia a una explicación para las interfaces para sistemas ACF explicando cómo deben ser implementadas y porque.

Los sistemas ACF predicen la afinidad de un usuario para un conjunto de elementos o información. A diferencia de los sistemas de filtrado de información basados en contenido, como aquellos

desarrollados utilizando recuperación de la información o tecnologías de inteligencia artificial, la decisión de filtrado en ACF es basada en análisis humano no en análisis por máquina del contenido. Cada usuario en un sistema ACF califica elementos que ellos han experimentado, con el objetivo de establecer un perfil de intereses. Entonces los sistemas ACF emparejan usuarios con personas de intereses y gustos similares. Por lo tanto las calificaciones de personas similares son usadas para generar recomendaciones para el usuario.

ACF ha tenido varias ventajas significativas sobre los tradicionales filtros basados en contenido, principalmente porque no dependen del error generado por el análisis de contenido por máquinas. Las ventajas incluyen la habilidad para filtrar cualquier tipo de contenido, por ejemplo, textos, arte, música, etc. La habilidad para filtrar en basa a conceptos complejos y difíciles de representar como gusto y calidad, y la habilidad de hacer recomendaciones fortuitas.

Es importante notar que las tecnologías ACF no necesariamente compiten con aquellas que filtran en pase a contenido. En la mayoría de los casos ellos pueden ser integrados para brindar una poderosa solución hibrida.

A pesar del gran éxito de las tecnologías ACF con sistemas colaborativos automáticos de filtrado en dominios del entretenimiento en dominios de alto riesgo no lo han sido. Existen varias razones por las cuales los sistemas ACF no son confiables. Primero los sistemas ACF son procesos estocásticos que calculan predicciones basado en modelos que son aproximaciones heurísticas a los procesos humanos. Segundo, y probablemente lo más importante, los sistemas ACF basan sus cálculos en datos extremadamente dispersos y datos incompletos. Es por eso que los sistemas ACF por lo general dan buenas recomendaciones pero ocasionalmente dan un pésimo resultado.

En algunas ocasiones los sistemas ACF son vistos como cajas negras que no pueden ser cuestionadas y no se le dan indicadores que le permitan determinar si la recomendación es confiable o no. Explicaciones de las recomendaciones otorgadas por el sistema proveen mayor confianza hacia estos sistemas.

Fuentes de error

Las fuentes de error en los sistemas ACF pueden ser agrupadas en dos tipos:

- **Errores de modelo o de proceso**: Estos ocurren cuando los sistemas ACF usan procesos para calcular recomendaciones que con concuerdan con los requerimientos del usuario.
- **Errores de datos**: resultan de que los datos usados para el cálculo de la recomendación este tipo de errores puede ser clasificado en:
    - Información insuficiente.
    - Información incorrecta o de muy mala calidad.
    - Datos de gran variación.

La falta de información e información dispersa son características innatas de este estilo de problemas si se tuvieran los datos completos no existiría necesidad de estos sistemas. También cuando se agrega un nuevo elemento, por ejemplo, cuando hay una nueva película casi no hay registros de ella y poco a poco los usuarios la califican. Por otro lado se puede no tener información completa, por ejemplo, si un usuario es nuevo no se tiene información de él y por lo tanto no se tiene un perfil en base al cual recomendarle elementos. A pesar de que se tenga cantidades

considerables de información acerca de los usuarios y elementos existe la posibilidad de error por datos que contengan errores, por ejemplo, se puede calificar por accidente alguna película la cual se tiene una perspectiva totalmente diferente a la que realmente se tiene.

Explicaciones

Dar explicaciones al usuario es un mecanismo para poder manejar los errores de recomendación ya que brinda elementos para que el usuario pueda determinar si la recomendación que se le dio es correcta o no. Esta idea es natural para los humanos, cuando recibimos alguna sugerencia de otra persona nosotros la reconocemos como recomendaciones imperfectos y en base a la explicación de la recomendación tomamos la decisión de tomar o no tomar dicha recomendación y si la persona que nos recomienda algo es de confianza solemos seguir su recomendación excepto cuando esa recomendación no tiene sentido y en ese caso hacemos la pregunta ¿Por qué?

Algunos de los beneficios de otorgar explicaciones en sistemas de recomendación son:

- Justificación: Entender el razonamiento detrás de una recomendación ayuda a decidir cuanta confianza darle a la recomendación.
- Participación de los usuarios: la participación del usuario en el proceso de recomendación permite que el usuario pueda agregar su conocimiento al proceso.
- Educación: educación al usuario acerca del proceso de recomendación permite hacerle entender cuáles son los alcances y limitaciones del sistema.
- Aceptación: permite que el usuario acepte el sistema sabiendo sus alcances y limitaciones.

Los modelos de explicación de recomendaciones se pueden clasificar en 3 tipos principales:

- Modelos de caja blanca: estos modelos otorgan el beneficio de que el usuario tiene una noción del proceso del cual se calculó la recomendación dada. Estos modelos por lo general contienen los siguientes pasos.
    - Los usuarios introducen sus calificaciones mediantes un perfil
    - Los sistemas ACF localizan personas con perfiles similares (vecinos)
    - Las calificaciones de los vecinos son combinadas en forma de recomendaciones.
- Modelos de caja negra: Estos modelos se caracterizan por que no existe forma en la cual los usuarios puedan transmitir el proceso de recomendación.
- Modelos de desinformación conceptual: Estos modelos no explican toda la complejidad del sistema de recomendación, solo le muestra al usuario una parte del proceso.

Modelos de recomendación de problemas

Los diversos jueces en línea no han creado sistemas de recomendación personalizados, la gran mayoría muestra sus problemas ordenados de acuerdo con alguna de las siguientes características:

- Listado de problemas ordenado según su fecha de creación.
- Listado ordenado de acuerdo a su dificultad asignada por un experto.
- Listado ordenado de acuerdo a concursos que realizaron los problemas.
- Listado ordenado de acuerdo a la razón de envíos aceptados y envíos hechos.
- Búsqueda de acuerdo al nombre o identificador del problema.

Capítulo 3: Diseño del modelo de recomendación

Procesos de resolución de problemas en concursos de programación

En la olimpiada de informática se busca que los alumnos tengan la capacidad de análisis, abstracción y algoritmos para la resolución de problemas que involucran grandes cantidades de datos no solamente las habilidades para programar, como Frank Robson comenta en la Olimpiada Internacional de Informática en 2013 [27] “Para poder resolver estos problemas no solo se necesita ser un programados, que es poder decirle a la computadora que hacer, sino también ser codificadores creativos, quienes pueden imaginar que es lo que se le necesita decir a la computadora que hacer, lo cual no es la programación sino las matemáticas detrás de eso”.

Para poder resolver estos problemas se requiere no solo dominar los algoritmos, sino entender su esencia y adaptarlos de acuerdo a nuevas problemática, ya que la olimpiada busca siempre proponer problemas inéditos.

En México se han analizado diversas formas de entrenamiento y las que se han identificados como exitosas son aquellas que se basan en el trabajo constante, al mismo tiempo que este trabajo sea en el límite de las habilidades, es decir, se busca que el alumno constantemente trabaje con problemas retadores.

Una de las delegaciones de la Olimpiada de Informática en Mexico que constantemente tiene alumnos dentro de la selección nacional es el Distrito Federal y estado de México, desde el 2008 a la fecha (hasta la IOI del 2015) esta delegación ha enviado 13 veces a un integrante de alguna de esta delegación a la olimpiada internacional de las cuales 2 ganaron medalla de bronce y una medalla de plata. Esta delegación atribuye su éxito a la utilización de una metodología al momento de resolver problemas.

Metodología de resolución de problemas

La metodología para la resolución de problemas del Distrito Federal y Estados de México, se basa en una serie de pasos que buscan asegurar acercarse a la solución del problema evitando la aparición de errores en etapas cercanas a la conclusión del proceso, ya que estos errores tienden a ser muy costosos en tiempo. Esto esta metodología es muy similar a los ciclos de vida del desarrollo del software por lo cual puede adaptar diferentes conceptos de esta área.

A continuación se muestra un diagrama de la metodología utilizados por el Distrito Federal y Estado de Mexico.

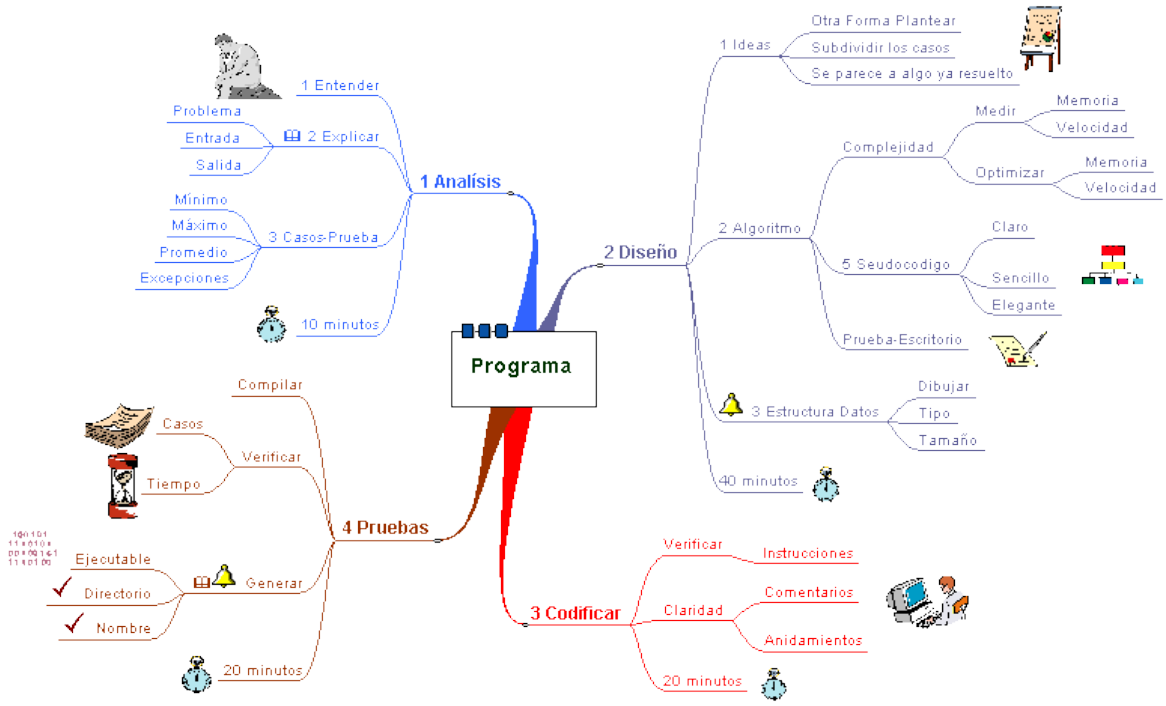


Ilustración 3 Metodología Distrito Federal y Estado de México

Debido a que el alumno necesita resolver 3 o 4 problemas en un examen de 5 horas es muy importante medir la velocidad del alumno.

Esta metodología está dividida en 4 Etapas principales:

1) Análisis: Durante esta etapa se busca comprender las características del problema, así como sus limitantes y forma de interactuar con el evaluador. Se recomienda que el tiempo otorgado a este paso sea de alrededor 10 min, pero en ocasiones es necesario otorgar más tiempo para no descuidar el entendimiento. Se subdivide en 3 etapas.
    a. Entender: el alumno debe comprender que es lo que se le está pidiendo.
    b. Explicar: Es común que se tenga la noción de entender cuando realmente no se entendió completamente la idea, se recomienda el proceso de explicar a alguien más (durante un examen imaginarse explicando a alguien más), esto resulta de gran ayuda para descubrir detalles que no se hayan entendido. Se le pide al alumno que preste atención en las siguientes características:
        i. Problema.
        ii. Entrada.
        iii. Salida.
    c. Casos de prueba: Es muy importante que antes de generar una idea de solución se comprenda las limitaciones, es por ello, que se recomienda pensar en las siguientes casos:
        i. Mínimo.
        ii. Máximo.
        iii. Promedio.

iv. Excepciones.

2) Diseño: Esta es la etapa en donde el alumno demuestra sus habilidades de análisis, durante esta etapa el alumno debe tener la capacidad de generar diferentes soluciones y al mismo tiempo identificar si esas soluciones son capaces de satisfacer todas las necesidades del problema.
    a. Ideas: Se trata de generar un esquema general de como un problema va a ser resuelto y una estimación de la complejidad del problema en tiempo y memoria para determinar si es viable pasar a codificar la solución. En ocasiones la solución no es obvia por lo tanto es necesario:
        i. Buscar otra forma de plantear el problema.
        ii. Subdividir los casos.
        iii. Identificar similitudes con problemas resueltos previamente.
    b. Algoritmos: Se tiene que describir cuales van a ser los pasos de la solución lo suficientemente detallado para que no exista duda de la posibilidad de implementar la solución. Se pide poner atención a lo siguiente:
        i. Complejidad
            1. Medir memoria y velocidad.
            2. Optimizar memoria y velocidad.
        ii. Seudocódigo
            1. Claro.
            2. Sencillo.
            3. Elegante.
        iii. Prueba de escritorio
    c. Estructura de datos: Es común que algunos algoritmos requieran la utilización de alguna estructura de datos que facilite el análisis de los datos es por ello que el alumno debe identificar que característica deben tener las estructuras de datos necesarias y para poder hacer esto se recomienda:
        i. Dibujar.
        ii. Identificar tipo de estructura de datos.
        iii. Tener claro el tamaño de la estructura de datos.
3) Codificar: En esta etapa el alumno muestra sus habilidades técnicas para consolidar sus ideas en código que la maquina pueda verificar, esta es una de las etapas que depende mucho de la practica previa, ya que esta facilita la transferencia de idea a código.
    a. Verificar instrucciones
    b. Claridad
        i. Comentarios
        ii. Anidamientos
4) Pruebas: Esta etapa final sirve para verificar que el código generado cumpla con las características que se planteó, se recomienda probar con al menos los casos que se idearon en la sección de análisis.
    a. Compilar
    b. Verificar
        i. Casos
        ii. Tiempo

c. Generar código

Lista de temas que se evalúan en la olimpiada de informática

La Olimpiada Internacional de informática sigue un programa de estudios conocido como "IOI Syllabus" [28], en este se mencionan los siguientes temas:

- Aritmética
- Geometría
- Estructuras Discretas
  - Lógica Básica
  - Técnicas de demostración
  - Conteo Básico
  - Gráficas y arboles
- Probabilidad Discreta
- Fundamentos de programación
  - Construcciones fundamentales de programación
  - Algoritmos y resolución de problemas
  - Estructura de datos fundamentales
  - Recursión
- Algoritmos y complejidad
  - Base de algoritmos
  - Estrategias algorítmicas
  - Estructuras de datos
  - Autómatas y complejidad
  - Algoritmos Avanzados
  - Algoritmos Geométricos

Diseño del modelo de simulación

Simulación

Para poder evaluar el comportamiento de los usuarios, con los diferentes modelos de recomendación, se ha diseñado una simulación sobre la cual los diferentes recomendadores podrán dar recomendaciones a los usuarios simulados.

Para poder explicar el proceso de recomendación se creó el diagrama de secuencia del proceso de simulación (Ilustración 4), en este proceso se contemplan tres entidades: Simulador, Recomendador y Usuario.

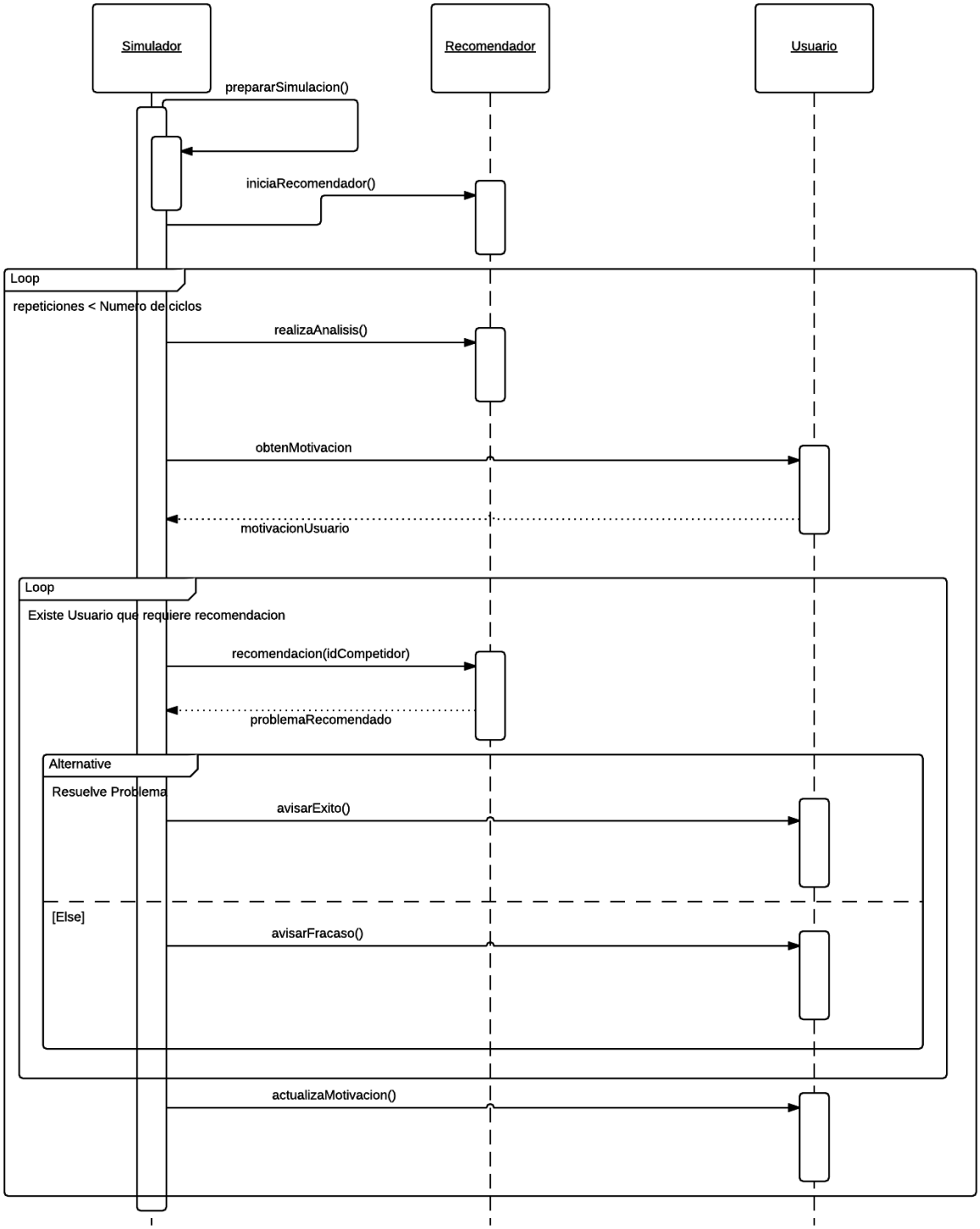


Ilustración 4 Diagrama de secuencia del proceso de simulación

La entidad Simulador es la encargada de coordinar a las otras dos entidades, determina la ejecución de los diferentes periodos, decide el orden en el cual los usuarios reciben sus respectivas recomendaciones, toma la decisión si se considera que el usuario puede resolver un problema y al igual que decide cuándo, una vez resuelto un problema, se considera que el usuario incrementa su habilidad.

El proceso de la simulación es el siguiente:

Se prepara la simulación: durante este periodo se borran los usuarios ficticios de la simulación anterior, se registra la nueva simulación y se crean los nuevos usuarios ficticios de la simulación. Después se llama a la función iniciaRecomendador().

Se repite $nCiclos$ veces la simulación de un ciclo.

Durante la simulación de cada ciclo, primero se llama a la función IniciaCiclo() del recomendador, después se agrega a una estructura de datos los usuarios que tienen una motivación de al menos 1.

Mientras la estructura de datos tenga algún usuario que requiera una recomendación se selecciona a algún usuario de ese conjunto al azar, se le pide al recomendador generar una recomendación para el usuario seleccionado.

Se determina si el usuario puede resolver el problema recomendado, si este es resuelto, se determina si el usuario sube de nivel.

Después de lo anterior se le informa al usuario cuál fue el resultado, de esta manera puede saber cuántos problemas resolvió y cuantos problemas no pudo resolver para determinar la nueva motivación.

Al terminar de dar las recomendaciones se actualizan las motivaciones de cada uno de los usuarios y se inicia con el siguiente ciclo.

Descripción de la estructura de datos selección de usuarios

Para poder seleccionar al azar al siguiente usuario que va a recibir una recomendación se necesitó una estructura de datos que pueda elegir al azar de un usuario de un conjunto de usuarios que tienen al menos una recomendación pendiente, esta estructura de datos soporta las siguientes funciones:

- agrega(usuario,r): agrega al conjunto de datos al usuario, indicando que este usuario debe ser seleccionado r veces.
- saca(): selecciona al azar a algún usuario.
- empty(): responde si existe algún usuario pendiente por seleccionar.
- cuantosRestantes(): responde cuantas recomendaciones están pendientes.
- quita(usuario): elimina al usuario sin importar cuantas veces falta ser seleccionado.

A continuación se muestra el código de la estructura de datos en el lenguaje C#.NET:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Simulacion
{
    class SelectorRandom
    {
```

```
Dictionary<int, int> posicion;
Dictionary<int, int> cantidad;
Random rnd;
int[] ids;
int n;
int total;
public SelectorRandom(int nElem)
{
    rnd = new Random(12492);
    ids = new int[nElem];
    n = 0;
    total = 0;
    posicion = new Dictionary<int, int>();
    cantidad = new Dictionary<int, int>();
}
public void agrega(int id, int cuantos)
{
    if (cuantos > 0)
    {
        ids[n] = id;
        cantidad[id] = cuantos;
        posicion[id] = n;
        n++;
        total += cuantos;
    }
}
public int saca()
{
    int pos = rnd.Next(n);
    int result = ids[pos];
    cantidad[result]--;
    if (cantidad[result] <= 0)
    {
        ids[pos] = ids[n - 1];
        posicion[ids[pos]] = pos;
        n--;
    }
    total--;
    return result;
}
public int cuantosDiferentes()
{
    return n;
}
public bool empty()
{
    return cuantosDiferentes() == 0;
}
```

```
        public int cuantosRestantes()
        {
            return total;
        }
        public void quita(int id)
        {
            if (!posicion.ContainsKey(id)) { return; }
            if (cantidad[id] == 0) return;
            n--;
            total -= cantidad[id];
            ids[posicion[id]] = ids[n];
        }
    }
}
```

Esta estructura se basa en 5 variables:

- n: el número de diferentes usuarios en la estructura.
- total: el número de recomendaciones pendientes.
- ids: es un arreglo en la cual de la posición 0 a la n-1 se encuentra cada uno de los identificadores de los usuarios para dar recomendación.
- posicion: es un mapeo del identificador del usuario a un entero, esté entero indica la posición en el arreglo ids del usuario.
- cantidad: es un mapeo del identificador del usuario a un entero, este entero indica cuantas recomendaciones están pendientes para el usuario.

Esta estructura de datos tiene la ventaja de que cada una de sus funciones son ejecutadas en tiempo constante o constante amortizado.

Modelo del problema

Se decidió que el modelo del problema no variara a lo largo del tiempo, por lo tanto no cuenta con un estado definido.

Cada problema pertenecerá a solo un tema, es decir que Q será una partición de P. Cada problema tendrá asignada una dificultad. Para poder determinar si un usuario puede resolver un problema se utiliza $P(resolver = 1|N \leq n)$ y para determinar si se incrementa de nivel a un alumno se utiliza $P(N > n|resuelto = 1)$.

Parámetros

- $tema \in Q$
- $dificultad \in \mathbb{N},\ dificultad\ \leq Nivel\ máximo$
- $P(resolver = 1|N \leq n), resolver\ \in \{0,1\}, n \in \mathbb{N}$
- $P(N > n|resuelto = 1), n \in \mathbb{N}, resuelto \in \{0,1\}$

Modelo del usuario

Para poder simular a los usuarios se creó el siguiente modelo:

Estado

- m: motivación, $m \in [0, motivación\ máxima)$: Indica cuantos problemas el usuario intentará en el ciclo en el que se encuentra.
- $\vec{n}$: habilidades, $\vec{n} \in \mathbb{N}^{|Q|}$, $\forall_{i \in [1,2,\dots,|Q|]} n_i \leq Nivel\ máximo$: un vector de números naturales que indica el nivel del usuario para cada tema.
- $fallados \in \mathbb{N}$: Indica el número de problemas fallados anteriormente.
- $resueltos \in \mathbb{N}$: Indica el número de problemas resueltos anteriormente.
- $incMotivación \in (incremento\ mínimo, incremento\ máximo)$: indica el incremento en la motivación que el usuario tendrá al terminar el ciclo.

Parámetros

- $aPositiva \in [0, 2]$
- $aNegativa \in [0, 2]$
- $fFacilidad \in [0, 2]$
- $motivaciónInicial \in [1, 7]$

Eventos

- Resolvió problema(p)
  - Si dificultad(p) > $n_{tema(p)}$
    - $incMotivación \coloneqq incMotivación + aPositiva * e^{-aPositiva * resueltos}$
  - Sino
    - $factor \coloneqq e^{(dificultad(p) - n_{tema(p)}) * fFacilidad}$
    - $incMotivación \coloneqq factor * aPositiva * e^{-aPositiva * resueltos}$
  - $resueltos \coloneqq resueltos + 1$
- Falló problema(p)
  - $incMotivación \coloneqq incMotivación\ - aNegativa * e^{-aNegativa * fallados}$
  - $fallados \coloneqq fallados + 1$
- Sube nivel (tema)
  - $n_{tema} \coloneqq\ n_{tema} + 1$
- Fin de ciclo
  - $m \coloneqq m + incMotivación$
  - $fallados \coloneqq 0$
  - $resueltos \coloneqq 0$

Inicio

- $m \coloneqq motivaciónInicial$
- $\vec{n} \coloneqq (0,0, \dots)$
- $fallados \coloneqq 0$
- $resueltos \coloneqq 0$

Fin

- Ciclos de simulación terminados
- $m < 1$

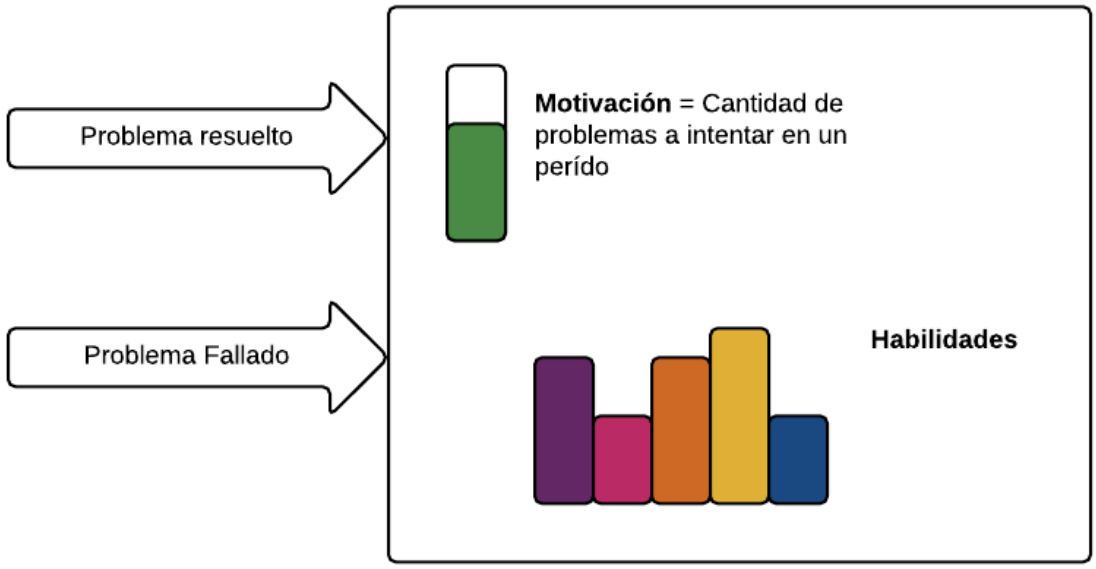


Ilustración 5 Modelo de Usuario

En este modelo se tiene como variables:

- Motivación: que indica el número de problemas que el usuario está dispuesto a intentar en un ciclo de simulación.
- Habilidades: Es un grupo de variables enteras también nombradas como niveles, cada una corresponde a un tema, de tal manera que para cada tema cada usuario tiene su propio nivel.

La motivación y habilidades del usuario solo pueden ser afectadas por las actividades que realice el usuario, es decir, solo es afectado cuando se resuelve un problema, cuando se falla un problema o no se obtiene una recomendación.

Se considera que al fallar un problema la motivación disminuye, pero que tan rápido disminuye depende de las características del usuario y cuantos problemas se han fallado previamente, por ejemplo si la primera vez que se falla un problema la motivación puede disminuir 0.5 puntos, la segunda vez tendría que disminuir pero en menor medida, es decir, algo menor que 0.5 pero que no sea negativo. Por esas razones se optó por una función exponencial. Y esta puede ser descrita por una variable que nos indique cual va a ser la disminución en el primer fallo. A continuación muestro la ecuación para determinar el decremento cuando el usuario no resuelve el problema recomendado.

$$Decremento\ en\ Motivación = aNegativa\ *\ e^{-aNegativa*\#porblemasFalladosPreviamente}$$

La decisión de tomar la función de la distribución de exponencial fue basada en mi experiencia en la olimpiada de informática, donde he podido observar que siempre que un alumno resuelve un problema su motivación incrementa, pero este incremento disminuye entre más problemas haya resuelto previamente, es decir, necesitábamos una función que es positiva, su primera derivada sea positiva y su segunda derivada sea negativa, este

comportamiento ha sido observado en varios estudios como los realizados por Stanley Smith Stevens [29] en la cual habla como diferentes la percepción del cerebro a un estímulo externo comúnmente sigue una función de potencia.

A continuación en la Ilustración 6 se muestra la gráfica de esta ecuación con el valor aNegativa de 0.25, este es el valor utilizado durante las simulaciones.

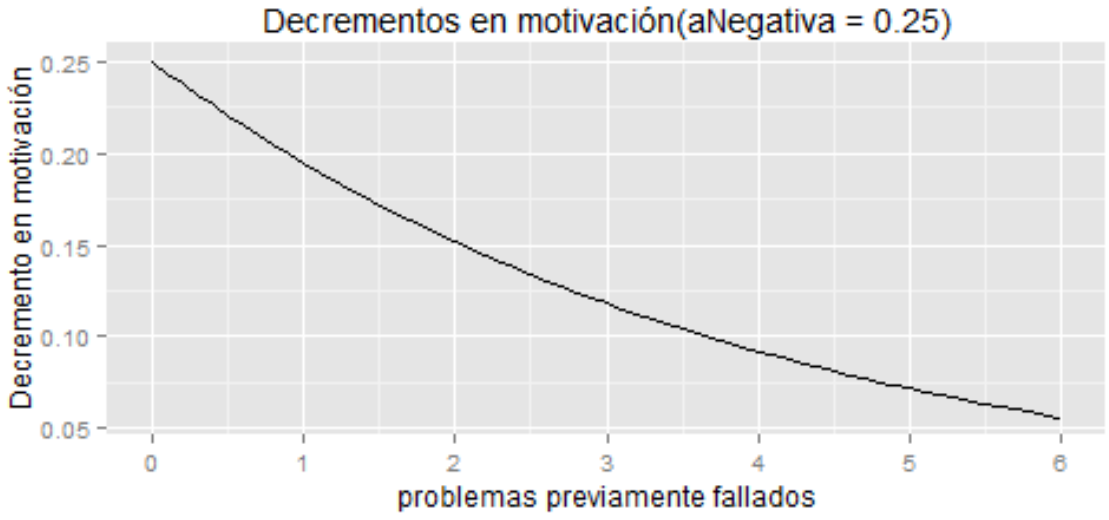


Ilustración 6 Gráfica decremento por problema fallado

El decremento total en cada ciclo de la motivación es la suma de cada uno de los decrementos causados por los problemas fallados en el ciclo.

En caso de tener éxito al momento de resolver un problema se tiene una afectación a la motivación de manera similar que en el caso de fallar, pero con la diferencia de que este incrementa la motivación. Con el parámetro aPositiva indicamos que tanto afecta al usuario que resuelva correctamente un problema.

$$Incremento\ en\ Motivación = aPositiva * e^{-aPositiva * \#porblemasResueltosPreviamente}$$

Por razones similares al decremento de la motivación se seleccionó la función de la distribución exponencial para modelar el incremento en la motivación. En el caso de resolver un problema puede ocurrir que la diferencia entre la dificultad de un problema y el nivel de habilidad de un usuario difiere mucho, en este caso el resolver un problema no le ofrece al usuario un incremento significativo en la motivación. Para modelar este comportamiento se agregó el factor de incremento que depende de la diferencia en dificultad.

Para las simulaciones se consideró que el valor de aPositiva es de 0.5, a continuación se muestra la gráfica del incremento de la motivación.

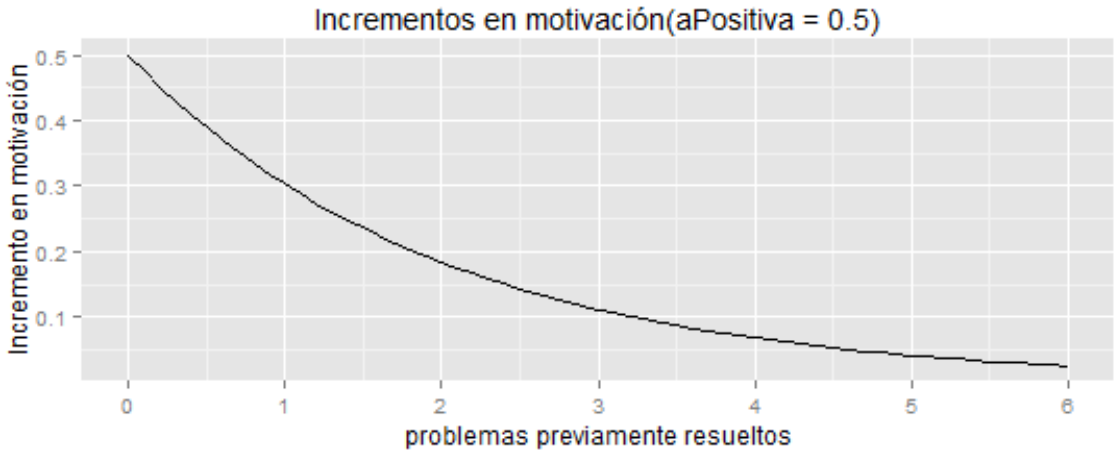


Ilustración 7 Gráfica incrementos en motivación

Para el caso de los problemas resueltos, también importa la diferencia del nivel del problema y la habilidad que tenga el usuario, es decir, si el usuario resuelve un problema muy difícil el incremento será el calculado previamente pero si es muy fácil solo se obtendrá un porcentaje del incremento de la motivación. Para modelar este fenómeno se creó el factor de incremento que es un valor entre 0 y 1 que determina que porcentaje del incremento de la motivación se tomara. Este factor es igual a 1 si el problema es de una dificultad mayor o igual al nivel que tiene el usuario, pero si la dificultad es menor este porcentaje disminuye entre mayor sea la diferencia. La ecuación del factor de incremento se decidió definir como lo siguiente:

$$Factor\ de\ Incremento = e^{difNivel * fFacilidad}$$

Donde

$$difNivel = \begin{Bmatrix} 0 & si\ nivelProblema - hablidadTema > 0 \\ nivelProblema - habilidadTema & en\ otro\ caso \end{Bmatrix}$$

Para las simulaciones se utilizó el valor de 1.25, a continuación se muestra la gráfica de la ecuación con estos valores.

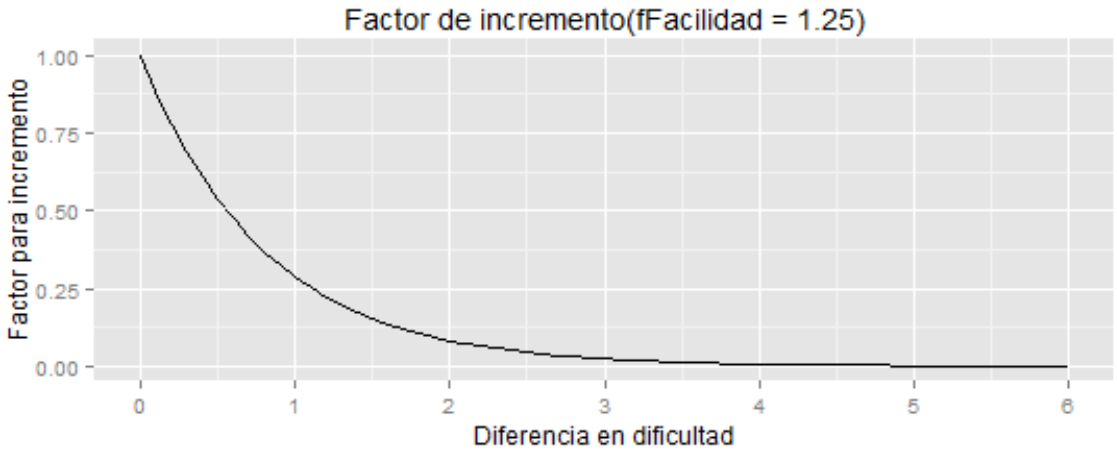


Ilustración 8 Gráfica de factor de incremento

De esta manera el incremento en la motivación por problema en un ciclo está dado por la siguiente ecuación:

$$Incremento\ por\ exito = Factor\ de\ Incremento * Incremento\ en\ Motivación$$

El incremento total en cada ciclo es igual a la suma de todos los incrementos individuales de los problemas resueltos con éxito.

A partir de lo anterior tenemos que las variables que definen a un usuario son:

- $fFacilidad$: Factor de facilidad, determina que tan importante es el impacto de resolver problemas más fáciles que el nivel del usuario.
- $aPositiva$: Afectación positiva, que determina que tan rápido el usuario deja de motivarse al resolver más problemas.
- $aNegativa$: Afectación negativa, que determina que tan rápido el usuario de desensibiliza a las fallas.
- $Motivación\ inicial$: determina la motivación con la cual el usuario inicia.

Identificación de nivel de usuarios en cada momento

Para poder identificar qué nivel tiene el usuario en cada momento se obtiene lo que es la historia del usuario. La historia del usuario es una secuencia de problemas que ha resuelto el usuario y que están ordenados de acuerdo al orden con el cual fueron resueltos. El usuario tiene diferentes habilidades en cada tema para calcular cada una se utiliza la historia de los problemas correspondientes a cada tema.

Un usuario siempre inicia con el nivel 0, todos los problemas están etiquetados manualmente con un nivel entre 1 y 5. El proceso para etiquetar el nivel de un usuario es el siguiente:

- Se toman los últimos tres problemas que resolvió el usuario.
- Se saca la mediana de la dificultad de esos tres problemas.

- El nivel del usuario es el máximo entre el nivel que tenía en el tiempo t-1 y la mediana calculada.

Descripción método de evaluación de aceptación de problemas

Para poder determinar si un usuario u puede resolver un problema p, se calcula previamente la probabilidad de que se resuelva el problema p dado que se trata del usuario u, $P(p|u)$, como la información que se posee es muy dispersa se utiliza una tercer variable z que representa a un usuario con un nivel z de habilidad, de esta manera se puede tratar a un conjunto de usuarios como los mismos con el uso de la variable y de esta manera se cuenta con información suficiente para calcular las probabilidades necesarias.

Con lo anterior el problema se transforma en encontrar la probabilidad de que se resuelva el problema r dado que el nivel del usuario es z o menor $P(r|z)$. Esta probabilidad es calculada de la siguiente manera:

$$P(r|z) = \frac{|\{Usuarios\ que\ resolvieron\ r\ y\ tienen\ un\ nivel\ menor\ igual\ a\ z\}|}{|\{Usuarios\ con\ un\ nivel\ menor\ o\ igual\ a\ z\}|}$$

Como la habilidad de un usuario depende del área del conocimiento se creó una variable z para cada uno de los posibles temas, dependiendo del problema se sabe el tema y por lo tanto el valor z correspondiente.

La variable z puede adquirir los siguientes valores:

- -1: El usuario ya no resuelve más problemas debido a que se rindió o no tiene motivación para continuar.
- 0: Iniciando con tema.
- 1: Puede resolver problemas que se utilizan como ejemplos.
- 2: Puede resolver problemas que tienen ligeras modificaciones a los problemas ejemplo.
- 3: Puede resolver problemas que es necesario llevar a cabo un análisis sencillo para su solución.
- 4: Puede resolver problemas que requieren de un análisis minucioso para su resolución.
- 5: Puede resolver problemas más difíciles que las clasificaciones anteriores.
- 6: El usuario ha completado todos los problemas.

Cada uno de los problemas fue clasificado con alguno de los niveles anteriores, con el objetivo de ayudar al cálculo de las probabilidades.

Los temas utilizados durante la simulación son:

- Introducción
- Básico
- Recursión
- Recursión con parámetros
- Búsquedas
- Problemario

La decisión de que el usuario resuelve el problema p es una variable aleatoria con probabilidad de éxito $P(r|z)$.

Descripción método de evaluación de cambio de nivel

Para actualizar la variable z necesitamos poder cambiar su valor de acuerdo a los avances que el usuario haga, se consideran algunas reglas básicas: si la motivación del usuario es menor a 1 eso quiere decir que z se cambia a -1, si el usuario resuelve todos los problemas la variable z cambia a 6, un usuario nunca disminuye de nivel con excepción de las reglas anteriores y en cualquier otro caso el usuario sube de nivel solo cuando resuelve problemas.

Dado que un usuario resolvió un problema la probabilidad de que suba de nivel está determinada por la probabilidad que el nivel de un usuario sea mayor que z dado que resolvió el problema p, $P(N > z|p)$. Esta nueva probabilidad puede ser calculada de manera similar a $P(r|z)$.

La decisión de que el usuario sube de nivel está dada por una variable aleatoria con una probabilidad de éxito $P(N > z|p)$.

La ilustración 9 muestra el diagrama de estados de los usuarios, en el cual se puede observar cómo se comporta un usuario a lo largo de la simulación.

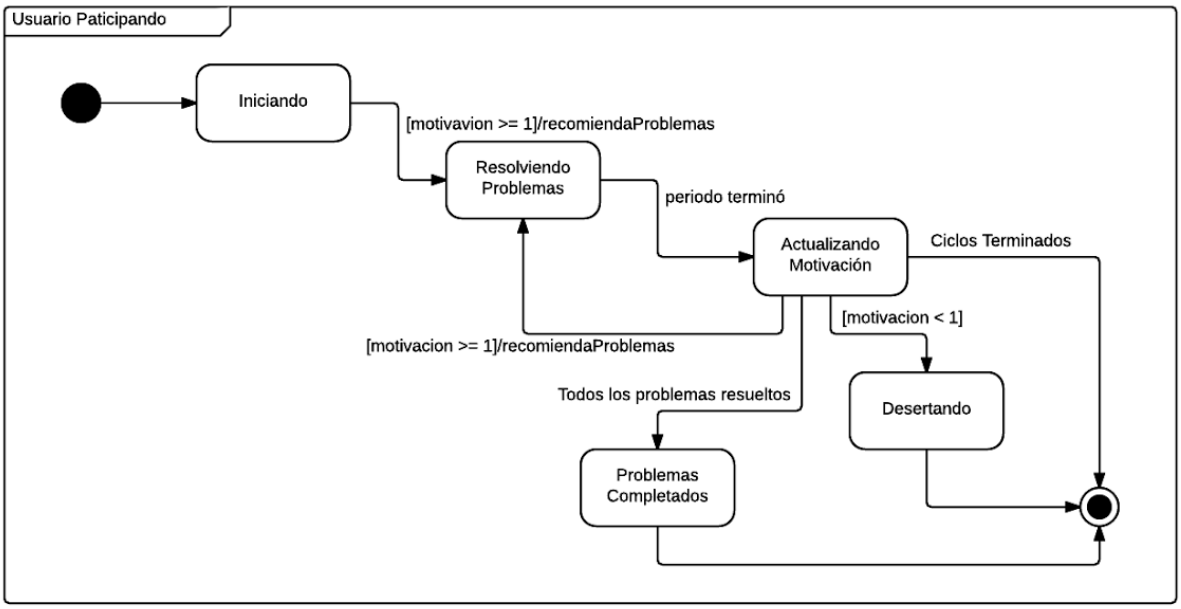


Ilustración 9 Diagrama de estados de los usuarios

El primer estado por el que pasa un usuario es el estado iniciando, donde el usuario es creado, se inicializan sus variables y se guarda en la base de datos sus características. Durante el estado resolviendo problemas el usuario está dentro del proceso de intentar problemas, resolverlos o fallarlos que se mencionaron previamente, al final de cada ciclo las motivación de los usuarios es actualizada y se determina si el usuario sigue en la simulación o si termina ya sea porque se terminaron los ciclos, su motivación es menor que 1 o completo todos los problemas.

Recomendaciones al azar

Este es el más sencillo de los modelos de recomendación es simplemente generar recomendaciones al azar. Lo único que necesita saber el recomendador es cuales problemas se pueden recomendar.

Cada uno de los recomendadores interactúa con el simulador por medio de tres funciones: IniciaRecomendador(), realizaAnalisis(), recomendacion(idCompetidor). Para el caso de este recomendador lo que tiene que realizar en cada una de esas funciones es:

- iniciaRecomendador(): carga la lista de problemas.
- realizaAnalisis(): No es necesario realizar un análisis previo.
- recomendacion(idCompetidor): Elegir al azar un problema.

Para el caso de este recomendador no hay ninguna variable que modifique el comportamiento del recomendador.

La complejidad de la función “realiza análisis” es constante así como la función “recomendación”.

Algorithm 1 Recomendaciones al azar

```
1:  procedure recomienda(u, P)    ▷ recomienda un problema al usuario u en base a los problemas P
2:      p ← eligeProblemaAlAzar(P)
3:      return p
4:  end procedure
5:  procedure rec_t(u, P)
6:      numRecomendaciones ← f_t(u)
7:      rec ← ∅
8:      for i = 1 to numRecomendaciones do
9:          rec ← rec ∪ recomienda(u, P)
10:     end for
11:     return rec
12: end procedure
```

Recomendaciones en base a un experto

Este modelo de recomendación es el utilizado actualmente en algunas de los jueces en línea, en aquellos que cuentan con un pequeño número problemas. Este modelo utiliza como base una lista ordenada de todos los problemas que existen y el recomendador utiliza esta lista para recomendar los problemas en ese orden. Una vez que un problema ha sido recomendado no se vuelve a recomendar durante t_{fuera} ciclos.

Para las funciones que se requieren el modelo se comporta de la siguiente forma:

- IniciaRecomendador(): limpia la tabla en la cual se guarda el momento en el cual se dio una recomendación, esto se guarda para identificar cuales problemas pueden ser recomendados, e inicializa las variables del recomendador.
- realizaAnalisis(): incrementa en uno el contador de ciclos de tiempo.

- recomendacion(idCompetidor): Selecciona el primer problema de la lista que no ha sido resuelto por el usuario y que tampoco se le ha recomendado durante los últimos t_{fuera} ciclos.

La única variable que modifica el comportamiento de este recomendador es la variable t_{fuera}, la cual indica por cuanto tiempo no se volverá a repetir una recomendación, esta variable fue creada para evitar que el recomendador se dedique a solo recomendar un problema que le pudiera ser difícil resolver al usuario.

El principal problema con este modelo es que no siempre se cuenta con la posibilidad de que alguien conozca todos los problemas, identifique su dificultad y genere la lista ordenada, especialmente en aquellos jueces en línea donde el número de problemas ascienda a miles.

La complejidad de la función "realiza análisis" es constantes mientras que la función "recomendación" toma tiempo $O(p)$ donde p es el número de problemas.

Algorithm 2 Recomendaciones basadas en un experto

procedure $recomienda(u, P, L, fc)$ ▷ recomienda un problmea al usuario u en base a la lista de problemas L y no repite recomendación en fc ciclos
  $A \leftarrow problemasResueltosPor(u)$
  $B \leftarrow \bigcup_{i=max(0,t-fc)}^{t-1} rec_i(u)$
  $E \leftarrow P \cap A^c \cap B^c$ ▷ problemas elegibles
  $p \leftarrow$ primer elemento $l_i \in L \wedge l_i \in E$
  return p
end procedure
procedure $rec_t(u, P, L, fc)$
  $numRecomendaciones \leftarrow f_t(u)$
  $rec \leftarrow \emptyset$
  for $i = 1$ **to** $numRecomendaciones$ **do**
    $rec \leftarrow rec \cup recomienda(u, P, L, fc)$
  end for
  return rec
end procedure

Recomendaciones basadas en el número de inversiones

Este modelo de recomendación utiliza como base el orden en el cual un usuario ha resuelto problemas, para así comparar cada usuario con los otros, identificando quienes son similares, y en base a eso identificar cuales problemas no ha resuelto el usuario pero que los usuarios similares si y dar como recomendación al problema que más se repita. Este algoritmo de recomendación fue creado para poder atacar a este problema.

El primer concepto a definir en este modelo es historia. Una historia de un usuario es una secuencia de problemas en la cual los problemas aparecen en el orden en que fueron resueltos, solo aparecen

aquellos problemas que el usuario a resuelto, es decir, que el usuario ha obtenido una puntuación del 100.

Dadas dos historias de usuarios diferentes U_1y U_2, donde $O_{u,p}$ es la posición del problema p en la secuencia del usuario u, los problemas a, b forman una inversión si: a es diferente de b, $O_{1,a} < O_{1,b}$ y $O_{2,a} > O_{2,b}$. De lo anterior podemos observar que el máximo número de inversiones es igual al número de parejas de problemas que es $\frac{n(n-1)}{2}$ donde n es el número de problemas que las dos historias tienen en común.

Para poder identificar qué problema se recomendara se necesitan tres características:

- Encontrar usuarios que sean similares al usuario actual.
- El criterio sobre el cual se basó para encontrar que son similares sea el más amplio posible.
- El número de problemas que no ha resuelto el usuario y que el otro usuario ha resuelto sea el máximo posible.

Para poder juntar estas tres características se creó una función llamada score, que evalúa que tan conveniente es tomar algún usuario como similar. A continuación se muestra la ecuación.

$$score(u_1, u_2) = similitud * |problemas\ compartidos| * complemento$$

Donde

$$complemento = |problemas\ de\ u_2| - |problemas\ compartidos|$$

$$similitud = 126 - \sqrt{2 * inversiones(u_1, u_2)}$$

Cada uno de los factores es un valor entre 0 y 126. Similitud indica que tan similares son las dos historias considerando las inversiones, $problemas\ compartidos$ es la intersección entre el conjunto de problemas en las dos historias, es decir, los problemas que se consideraran para calcular la similitud, y $complemento$ indica cuantos problemas ha resuelto el usuario 2 que el usuario 1 no ha resuelto.

Se sabe que entre más inversiones tengan las historias de dos usuarios más diferentes son, también que entre más elementos tengan en común es más grande el máximo de inversiones que se puede tener. Para reducir el efecto de cuadrático del número de inversiones se obtiene la raíz cuadrada del doble del número de inversiones, de esta manera el rango de este valor esta entre 0 y 126. Donde 126 quiere decir que existen todas las inversiones posibles y 0 que no hay inversiones. Por los tanto a 126 le restamos ese valor para que de esta manera se invierte el orden, es decir, 126 significa que no hay inversiones y 0 que hay muchas inversiones.

El complemento es simplemente la cardinalidad de los problemas que tiene la historia del usuario 2 menos la cardinalidad de la intersección entre las dos historias.

Para poder calcular el número de inversiones en tiempo $O(n \log n)$ se utilizó el algoritmo [30, p. 221] que se basa en la idea de utilizar mergesort como base para identificar cuantas inversiones existen en una permutación, se considera como el orden original la historia del usuario 1 y como la permutación la historia del usuario 2.

Una vez que se tiene la medida score, el modelo de recomendación sigue los siguientes pasos:

1) Se eligen a los topN usuarios con el mayor score, este grupo será nombrado usuarios similares.
2) Se hace la unión de los problemas en las historias de los usuarios similares.
3) Se eliminan los problemas que ya se solucionaron o que se han recomendado en los últimos t_{fuera} ciclos.
4) Se elige aquel problema que aparece en la mayor cantidad de historias, en caso de empate se selecciona el que aparece primero en la historia del usuario con mayor score.
5) En caso de no tener algún problema utiliza coldStart (Otro algoritmo de recomendación).

Por lo tanto este modelo de recomendación responderá de la siguiente forma a las funciones de un recomendador:

- IniciaRecomendador(): Limpia la tabla en la cual se guardan las recomendaciones anteriores e inicializa las variables necesarias.
- realizaAnalisis(): Calcula el parámetro score de cada pareja de usuarios, guarda en la base de datos los topN usuarios con mayor score para cada usuarios e incrementa el contador de tiempo.
- recomendacion(idCompetidor): Elige problema de acuerdo al algoritmo anterior.

Las variables que afectan a este algoritmo son:

- t_{fuera} : El número de ciclos en el cual no se volverá a recomendar un problema.
- $topN$: el número de usuarios que serán considerados como similares para dar la recomendación.
- Mínimo de Usuarios: el número mínimo de usuarios necesarios para considerar la recomendación como válida.
- coldStart: Recomendador a ser utilizado cuando no se tenga alguna posible recomendación.

La complejidad de la función “realiza análisis” es de $O(u^2 p \log p)$ donde u es el número de usuarios y p es el número de problemas. Mientras que la función “recomendación” toma un tiempo $O(mp)$ donde m es el máximo número de usuarios similares a considerar y p es el número de problemas.

Algorithm 3 Recomendaciones basadas en el número de inversiones

procedure $score(u_1, u_2)$
  $h_1 \leftarrow historia(u_1)$
  $h_2 \leftarrow historia(u_2)$
  $s \leftarrow \mid P \mid -\sqrt{2 * inversiones(h_1, h_2)}$
  $pc \leftarrow \mid h_1 \cap h_2 \mid$
  $c \leftarrow \mid h_2 \mid -pc$
  return $s * pc * c$
end procedure
procedure $recomienda(u, U, P, R, topN)$
  $A \leftarrow problemasResueltosPor(u)$
  $B \leftarrow \bigcup_{i=max(0,t-fc)}^{t-1} rec_i(u)$
  $E \leftarrow P \cap A^c \cap B^c$
  $Similares \leftarrow \emptyset$
  $Candidatos \leftarrow U$
  while $\mid Similares \mid < topN$ **do**
    $Similares \leftarrow Simulares \cup \text{argmax}_{x_i \in Candidatos}\, score(u, x_i)$
    $Candidatos \leftarrow U \cap Similares^c$
  end while
  $H \leftarrow historias(Similares)$
  $p \leftarrow \text{argmax}_{x_i \in P} \mid \{h_i | h_i \in H \wedge x_i \in h_i\} \mid$
  return p
end procedure
procedure $rec_t(u, U, P, R, topN)$
  $numRecomendaciones \leftarrow f_t(u)$
  $rec \leftarrow \emptyset$
  for $i = 1$ **to** $numRecomendaciones$ **do**
    $rec \leftarrow rec \cup recomienda(u, U, P, R)$
  end for
  return rec
end procedure

Recomendaciones basadas en similitud de usuarios

Este modelo se basa en la suposición que usuarios con resultados similares tendrán resultados similares, por lo tanto se podría encontrar que usuarios son similares, obtener el promedio de sus calificaciones y así poder predecir la calificación que podrá obtener el usuario. En base a las predicciones se puede elegir el problema que probablemente podrá ser resuelto por el usuario. Este algoritmo está basado en los filtros colaborativos basados en la similitud de usuarios.

El primer gran reto que se tuvo con este modelo fue plantear una métrica de similitud que logre identificar usuarios que tendrán calificaciones similares, para esto se utilizó el nivel identificado para cada uno de los usuarios.

$$sim(u_1, u_2) = 1 - \frac{\sum_{h \in habilidades} Abs(nivel(u_1, h) - nivel(u_2, h))}{\max(nivel) * |habilidades|}$$

También se cuenta con la información de las calificaciones obtenidas en cada uno de los problemas intentados, pero la similitud de las calificaciones agrupadas por temas representan la misma información que el nivel del usuario en cada tema y la motivación de los usuarios no representa la posibilidad de resolver algún problema.

Una vez calculada la similitud entre cada pareja de usuarios se puede guardar un subconjunto de ellos en una base de datos para así reducir el tiempo de respuesta de la recomendación. No es necesario guardar todas las similitudes, basta con guardar las similitudes más altas, pero se tiene que tomar en consideración que entre menor sea el número de similitudes guardadas mayores son las posibilidades de no poder tener predicciones para todos los problemas.

Una vez que se tienen las similitudes, se puede realizar la predicción de la calificación del usuario. A continuación se muestra la ecuación utilizada para realizar la predicción.

$uSimilares$ = {los mejores **top*N*** usuarios que intentaron p}

$$pred(u_1, p) = \frac{\sum_{u_2 \in uSimilares} calificacion(u_2, p) * sim(u_1, u_2)}{\sum_{u_2 \in uSimilares} sim(u_1, u_2)}$$

No se consideran las predicciones de problemas en los cuales $sim(u_1, u_2)$ sea menor que $similitud\ mínima$ o que el número de usuarios utilizados para dar la predicción sea menos que $Mínimo\ de\ usuarios$.

Para poder calcular la predicción primero se tienen que identificar un conjunto se usuarios que previamente intentaron resolver el problema p y que la similitud sea la máxima posible, al mismo tiempo, en vez de hacer un simple promedio sobre las calificaciones se puede utilizar la similitud entre usuarios como peso para realizar un promedio ponderador.

La serie de pasos que se tienen que realizar para dar una recomendación son los siguientes:

1) Buscar los usuarios similares, para acelerar este paso se guardaron en una base de datos las similitudes entre usuarios calculadas previamente.
2) Predecir calificaciones de los problemas en base a los usuarios similares.
3) Eliminar los problemas ya resueltos o que se han recomendado en los últimos t_{fuera} ciclos.
4) Elegir el problema con la mayor calificación predicha.
5) En caso de no tener alguna recomendación ya sea por falta de usuarios similares o falta de información del usuario, llamar al recomendador coldStart que es otro algoritmo de recomendación.

Este modelo responde a las funciones que se le piden a los recomendadores de la siguiente forma:

- iniciaRecomendador(): Limpia tablas utilizadas por el recomendador e inicializa la variables.
- realizaAnalisis(): Calcula las similitudes de cada pareja de usuarios y guarda en una base de datos las similitudes.
- recomendacion(idCompetidor): Se elige el problema de acuerdo al algoritmo previamente descrito.

Las variables que afectan el comportamiento de este modelo son:

- t_{fuera}: Indica por cuanto tiempo no se repetirá una recomendación.
- $coldStart$: Un recomendador que será utilizado cuando no se tenga una recomendación.
- $topN$: El número máximo de usuarios similares que será utilizado para calcular la predicción.
- $Mínimo\ de\ usuarios$: El número de usuarios necesarios para poder considerar como válida una predicción.
- $Similitud\ mínima$: la similitud mínima necesaria para considerarse como usuario similar.

La complejidad de la función "realiza análisis" es $O(u^2h)$ donde u es el número de usuarios y h es el número de habilidades. Mientras que la función "recomendación" es $O(mp)$ donde m es el máximo número de usuarios similares y p es el número de problemas.

Algorithm 4 Recomendaciones basadas en similitud de usuarios

procedure $pred(a, p)$
  $S \leftarrow usuariosSimilares(u) \cap usuariosIntentado(p)$
  return $\frac{\sum_{b \in S} r_{b,p} * sim(a,b)}{\sum_{b \in S} sim(a,b)}$
end procedure
procedure $recomienda(u, U, P, R, topN)$ ▷ recomienda un problmea al usuario en base a los usuarios similares
  $A \leftarrow problemasResueltosPor(u)$
  $B \leftarrow \bigcup_{i=max(0,t-fc)}^{t-1} rec_i(u)$
  $E \leftarrow P \cap A^c \cap B^c$
  $p \leftarrow \text{argmax}_{p \in E}\, pred(u, p)$
  return p
end procedure
procedure $rec_t(u, U, P, R, topN)$
  $numRecomendaciones \leftarrow f_t(u)$
  $rec \leftarrow \emptyset$
  for $i = 1$ **to** $numRecomendaciones$ **do**
    $rec \leftarrow rec \cup recomienda(u, U, P, R, topN)$
  end for
  return rec
end procedure

Recomendaciones basadas en similitud de problemas

Uno de los grandes problemas del modelo anterior (similitud entre usuarios) es que se tiene que calcular la similitud entre cada par de usuarios lo cual puede ser muy costoso en tiempo. Otra aproximación es utilizar la similitud entre los problemas, visto desde este punto un usuario obtendrá calificaciones similares en problemas similares, por lo tanto es posible utilizar los problemas que ha resuelto el usuario para predecir la calificación que obtendrá. Y como siempre el número de problemas es mucho menor que el número de usuarios encontrar la similitud entre problemas es menos costosa. Este algoritmo está basado en filtros colaborativos basados en la similitud de problemas.

Al igual que la similitud entre usuarios la similitud entre problemas se basa en las calificaciones de los usuarios, solo se utilizan para comparar aquellos usuarios que han intentado los dos problemas. A continuación se muestra la función completa para obtener la similitud entre dos problemas.

$$uComunes = \{\, usuarios\ que\ calificaron\ p_1\} \cap \{usuarios\ que\ calificaron\ p_2\}$$

$$\overrightarrow{P_1} =< calificacioes\ de\ p_1\ de\ los\ \boldsymbol{uComunes} >$$

$$\overrightarrow{P_2} =< calificacioes\ de\ p_2\ de\ los\ \boldsymbol{uComunes} >$$

$$simCalificaciones(p_1, p_2) = \frac{\sum_{u \in uComunes} \overrightarrow{P_1}[u] * \overrightarrow{P_2}[u]}{\left|\overrightarrow{P_1}\right| * \left|\overrightarrow{P_2}\right|}$$

Si $\left|\overrightarrow{P_1}\right|$ o $\left|\overrightarrow{P_2}\right|$ es cero la similitud es cero.

$$simDificultad(p_1, p_2) = 1 - \frac{Abs\big(nivel(p_1) - nivel(p_2, h)\big)}{\max(nivel)}$$

$$sim(p_1, p_2) = 0.8 * simCalificaciones(p_1, p_2) + 0.2 * simDificultad(p_1, p_2)$$

La similitud tiene dos componentes que son la similitud entre calificaciones y la similitud en la dificultad, se excluyó la similitud de tema debido a que esta similitud provoca que los usuarios solo resuelvan problemas de un mismo tema incrementando dramáticamente el número de veces que se tiene que llamar al recomendador auxiliar.

Debido a que son pocas las parejas de problemas es posible guardar todas las similitudes en una base de datos para así después utilizarlas al momento de dar una recomendación, de esta manera es posible no gastar tantos recursos calculando la similitud entre problemas cada vez que se da una recomendación.

El siguiente paso es predecir la calificación que obtendrá el usuario una vez que se saben las calificaciones de algunos problemas que ha intentado y las similitudes que se tienen entre los problemas. A continuación se muestra la función para poder predecir las calificaciones de los usuarios.

$$pSimilares = \{problemas\ resueltos\ por\ u\}$$

$$pred(u, p_1) = \frac{\sum_{p_2 \in pSimilares} calificacion(u, p_2) * sim(p_1, p_2)}{\sum_{p_2 \in pSimilares} sim(p_1, p_2)}$$

Se consideran problemas similares si la $sim(p_1, p_2) \geq minimaSimilitud$.

Solo se considera una predicción valida si se cuenta solo con al menos el $minimoProblemas$ similares.

Para poder predecir se consideran a todos los problemas que tienen una similitud de al menos $minimaSimilitud$, después se utiliza un promedio ponderado por la similitud con el problema que se está dando la recomendación.

Para poder realizar una recomendación se realizan los siguientes pasos:

1) Buscar problemas intentados previamente por el usuario.
2) Buscar problemas no resueltos y que no se han recomendado en los últimos t_{fuera} ciclos.
3) Predecir calificaciones de los problemas no resueltos, basándose en los resueltos.
4) Elegir el problema con la mayor predicción.
6) Si no se tiene recomendación llamar al recomendador coldStart que es otro algoritmo de recomendación.

Este modelo de recomendación responde a las funciones de la siguiente manera:

- iniciaRecomendador(): Limpia tablas utilizadas por el recomendador e inicializa variables.
- realizaAnalisis(): Calcula similitudes de cada pareja de problemas, incrementa el contador del tiempo y guarda en la base de datos las similitudes entre problemas.
- Recomendación(idCompetidor): Elige un problema de acuerdo al algoritmo anterior.

Las variables que afectan el comportamiento del algoritmo son:

- t_{fuera}: El número de ciclos en el cual una recomendación no se repetirá.
- $coldStart$: Recomendador que será utilizado en caso de que no se tenga recomendación.
- $minimo\ de\ usuarios$: el número mínimo de problemas que se deben de utilizar para considerar una predicción como correcta.
- $Similitud\ minima$: La similitud mínima que deben tener dos problemas para poderse contar en la predicción.

La complejidad de la función “realiza análisis” es $O(p^2u)$ donde p es el número de problemas y u es el número de usuarios. Mientras que la complejidad de la función recomendación es $O(p^2)$ donde p es el número de problemas.

Algorithm 5 Recomendaciones basadas en similitud de problemas

procedure $pred(a, p)$
    $S \leftarrow problemasIntentadosPor(u)$
    return $\frac{\sum_{b \in S} r_{u,b} * sim(p,b)}{\sum_{b \in S} sim(p,b)}$
end procedure
procedure $recomienda(u, U, P, R)$ ▷ recomienda un problmea al usuario u en base a los problemas similares
    $A \leftarrow problemasResueltosPor(u)$
    $B \leftarrow \bigcup_{i=max(0,t-fc)}^{t-1} rec_i(u)$
    $E \leftarrow P \cap A^c \cap B^c$
    $p \leftarrow \text{argmin}_{p \in E}\, pred(u, p)$
    return p
end procedure
procedure $rec_t(u, U, P, R)$
    $numRecomendaciones \leftarrow f_t(u)$
    $rec \leftarrow \emptyset$
    for $i = 1$ **to** $numRecomendaciones$ **do**
        $rec \leftarrow rec \cup recomienda(u, U, P, R)$
    end for
    return rec
end procedure

Recomendaciones basadas en factorización de matrices

Una de los mayores motivadores en los sistemas de recomendación fue el llamado "The Netflix Prize" [31], el cual prometió 1 millón de dólares a quien logre una mejora de la predicción de películas en un 10%, este premio fue obtenido por Simon Funk en 2006, quien utilizando factorización de matrices creo el algoritmo "FUNK SVD" [32]. Este algoritmo está basado en el algoritmo "FUNK SVD" modificado para poder responder al nuevo problema planteado.

La idea atrás de la factorización de matrices es convertir una matriz M de dimensiones $u\ x\ p$ en el producto de tres matrices M_u, I_f y M_p.

$$M = matriz\ de\ calificaciones\ usuarios\ x\ problemas$$

$$M_u = matriz\ de\ usuarios\ x\ features$$

$$M_p = matriz\ de\ problemas\ x\ features$$

$$I_f = matriz\ identidad\ de\ features\ x\ features$$

$$M = M_u * I_f * M_p^t$$

El número de features o características es determinado por quien implementa el algoritmo, entre más características es posible aproximarse más a la matriz original, lo cual permite tener control sobre uno de los problemas de los sistemas de recomendación que se llama sobreajuste el cual ocurre cuando las funciones de predicción se aproximan mucho a los datos de entrenamiento los cuales suelen contener ruido, por lo tanto tener sobreajuste incrementa la influencia de este ruido. Otra ventaja de este método es que se puede reducir la memoria necesaria para almacenar la matriz de calificaciones, también esta matriz tiene una nueva interpretación en donde cada columna de la matriz M_u representa una característica de un usuario y cada columna de M_p representa una característica del problema y los valores indican que tan importante es esa característica para el usuario o que importante es la característica en el problema respectivo.

"FUNK SDV" propone aproximar los valores de los factores basándose en el gradiente descendiente, la función que se busca minimizar es "Root Mean Square Error" que es la raíz cuadrada del error cuadrático medio, entre los valores predichos y la calificación real. Este método tiene la ventaja que solo es afectado por aquellas calificaciones que se han registrado y no es afectado por los valores faltantes en la matriz original como es el caso de la gran mayoría de los métodos de factorización de matrices que requieren estimar un aproximado de las calificaciones faltantes.

A continuación se muestra la ecuación de RMSE.

$$RMSE = \sqrt{\frac{\sum_{r \in R}(pred - r)^2}{|R|}}$$

$$pred(u,p) = \sum_{f \in Features} M_u(u,f) * M_p(p,f)$$

Si queremos encontrar el gradiente de RMSE podemos enfocarnos en el error creado por un solo feature para una predicción, lo cual nos lleva a querer encontrar la pendiente de la siguiente ecuación.

$$\left(M_u(u,f) * M_p(p,f) - r\right)^2$$

El gradiente con respecto M_u es:

$$\frac{\partial}{\partial M_u}\left(M_u(u,f) * M_p(p,f) - r\right)^2$$

$$= 2 * \left(M_u(u,f) * M_p(p,f) - r\right) \quad * \frac{\partial}{\partial M_u}\left(M_u(u,f) * M_p(p,f) - r\right)$$

$$= 2 * \left(M_u(u,f) * M_p(p,f) - r\right) \quad * M_p(p,f)$$

Definimos error de la siguiente manera

$$error = \ M_u(u,f) * M_p(p,f) - r$$

Por lo tanto el gradiente es

$$2 * error * M_p(p,f)$$

De igual forma para M_p:

$$\frac{\partial}{\partial M_p}\left(M_u(u,f) * M_p(p,f) - r\right)^2$$

$$= 2 * \left(M_u(u,f) * M_p(p,f) - r\right) \quad * \frac{\partial}{\partial M_p}\left(M_u(u,f) * M_p(p,f) - r\right)$$

$$= 2 * \left(M_u(u,f) * M_p(p,f) - r\right) \quad * M_u(u,f)$$

Definimos error de la siguiente manera

$$error = \ M_u(u,f) * M_p(p,f) - r$$

Por lo tanto el gradiente es

$$2 * error * M_u(u,f)$$

Para controlar la velocidad con la cual se aprende del error se crea el parámetro $lrate$, el cual es un factor que si es muy alto el algoritmo no converge a algún resultado y si es muy bajo el algoritmo converge muy lento. Con todo lo anterior para reducir el RMSE el proceso de entrenamiento para una puntuación r de la siguiente forma:

$$M_u(u,f) -= lrate * error * M_p(p,f)$$

$$M_p(p,f) -= lrate * error * M_u(u,f)$$

El algoritmo para encontrar las matrices M_u y M_p de tal manera que se reduzca RMSE es:

1) Inicializar las matrices M_u y M_p con los valores $\sqrt{\frac{50}{nFeatures}}$ de tal manera que la predicción para cada par de u, p sea 50.
2) Para cada característica f en $features$:
    a. Repetir hasta converger o el máximo número de iteraciones $nIteraciones$ sea superado:
        i. Para cada calificación registrada $r \in R$
            1. entrenar.

Una vez que se tienen las dos matrices es posible tener una predicción para cada pareja u, p.

Finalmente el proceso para dar una recomendación consiste en:

1) Obtener la predicción de cada problema que el usuario no ha intentado y que no han sido recomendado en los últimos t_{fuera}ciclos.
2) Elegir el problema con mayor predicción.

Este modelo de recomendación responde a las funciones de un recomendador de la siguiente forma:

- iniciaRecomendador(): Limpia tablas utilizadas por el recomendador e inicializa variables.
- realizaAnalisis(): Calcula las matrices M_u y M_p, incrementa el contador del tiempo y guarda en la base de datos las dos matrices.
- Recomendación(idCompetidor): Elige un problema de acuerdo al algoritmo anterior.

Las variables que afectan el comportamiento de este algoritmo son:

- t_{fuera}: El número de ciclos en el cual una recomendación no se repetirá.
- $coldStart$: Recomendador que será utilizado en caso de que no se tenga recomendación.
- $Número\ de\ características$: Número de características que se considerarán.
- $Taza\ de\ aprendizaje$: la velocidad con la cual el algoritmo aprende.
- $Mínimo\ de\ calificaciones$: Mínimo de calificaciones que el usuario debe tener para considerar las predicciones de calificaciones como aceptables.
- $Diferencia\ para\ considerar\ convergencia$: Cuando el RMSE difiere en menos de este valor con respecto a su valor anterior, se considera que las matrices M_u y M_p convergieron.
- $Máximo\ número\ de\ iteraciones$: Si se excede este número de veces el entrenamiento se deja de entrenar la característica.

La complejidad de la función "realiza análisis" es $O(nmk)$ donde n es el número máximo de iteraciones de entrenamiento, m es el número de calificaciones registradas y k es el número de características. Mientras que la complejidad de la función "recomendación" es $O(pk)$ donde p es el número de problemas y k es el número de características.

Algorithm 6 Recomendaciones basadas en la factorización de matrices

1: **procedure** $rmse(R, M_u, M_p)$
2: **return** $\sqrt{\frac{\sum_{r \in R} (pred(usurio(r), problema(r), M_u, M_p) - r)^2}{|R|}}$
3: **end procedure**
4: **procedure** $pred(a, p, M_u, M_p)$
5: **return** $\sum_{f \in Features} M_u(u, f) * M_p(p, f)$
6: **end procedure**
7: **procedure** $factorizaMatriz(R, Features, lrate, maxIteraciones)$
8: inicializa M_u y M_p con $\sqrt{\frac{50}{|Features|}}$
9: **while** rmse no converja e $iteraciones \leq maxIteraciones$ **do**
10: **for all** $f \in Features$ **do**
11: **for all** $r \in R$ **do**
12: $u \leftarrow usuario(r)$
13: $p \leftarrow problema(r)$
14: $error \leftarrow pred(u, p, M_u, M_p) - r$
15: $M_u(u, f) \leftarrow M_u(u, f) - lrate * error * M_p(p, f)$
16: $M_p(p, f) \leftarrow M_p(p, f) - lrate * error * M_u(u, f)$
17: **end for**
18: **end for**
19: **end while**
20: **end procedure**
21: **procedure** $recomienda(u, P, R, M_u, M_p)$ ▷ recomienda un problmea al usuario u en base a la factorización de matrices
22: $A \leftarrow problemasResueltosPor(u)$
23: $B \leftarrow \bigcup_{i=max(0,t-fc)}^{t-1} rec_i(u)$
24: $E \leftarrow P \cap A^c \cap B^c$
25: $p \leftarrow \text{argmin}_{p \in E}\, pred(u, p)$
26: **return** p
27: **end procedure**
28: **procedure** $rec_t(u, U, P, R, Features, lrate, maxIteraciones)$
29: $factorizaMatriz(R, Features, lrate, maxIteraciones)$
30: $numRecomendaciones \leftarrow f_t(u)$
31: $rec \leftarrow \emptyset$
32: **for** $i = 1$ **to** $numRecomendaciones$ **do**
33: $rec \leftarrow rec \cup recomienda(u)$
34: **end for**
35: **return** rec
36: **end procedure**

Capítulo 4: Implementación del modelo

Arquitectura del prototipo implantado

El prototipo implementado tiene tres objetivos principales: realizar simulaciones, mostrar gráficas de los resultados obtenidos de las simulaciones y proveer una interfaz gráfica. La implantación de esto se puede observar en la Ilustración 10, en ella se observan los tres componentes principales del sistema: simulación, gráficas e interfaz gráfica de usuario, junto con la base de datos de Karelotitlán y el componente externo que es R un sistema computacional para generar gráficas y estadísticas.

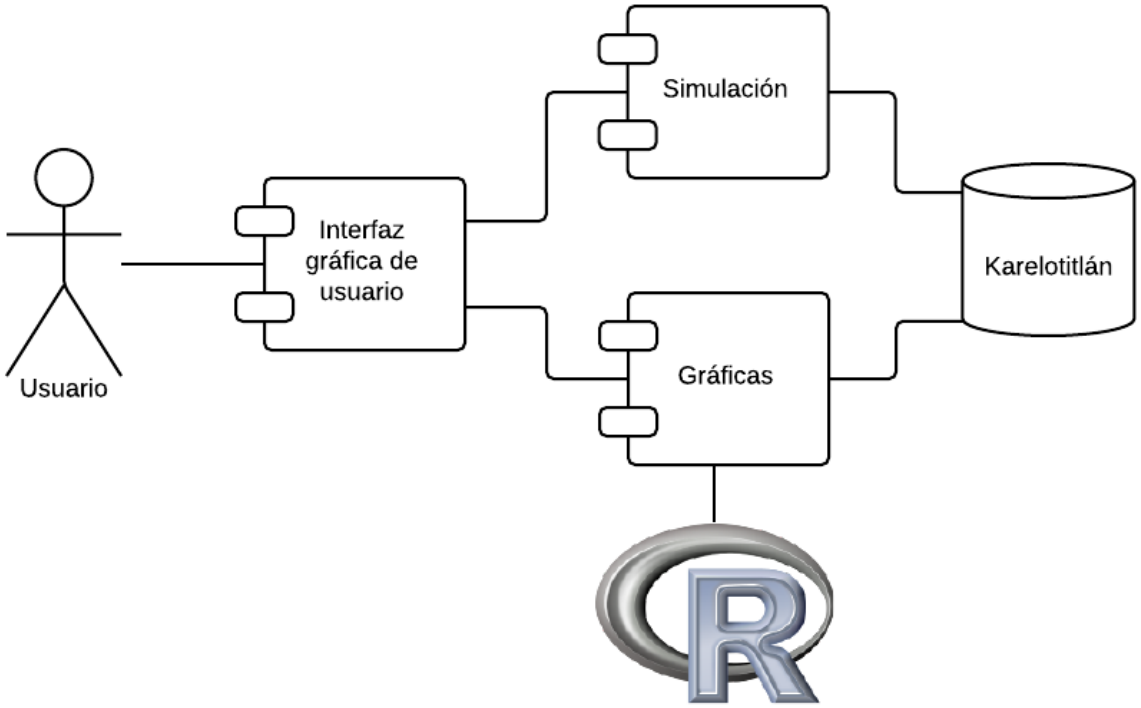


Ilustración 10 Diagrama general de prototipo

El usuario puede seleccionar desde la interfaz gráfica que modelo se utilizará para la simulación, cuantos ciclos se simularán, cuantos usuarios se crearán y en caso de requerirlo cual algoritmo se utilizará como $coldStart$. Durante la simulación el usuario puede observar el valor de las siguientes variables:

- Número de ciclos a completar.
- Número de ciclos completados.
- Número de recomendaciones a dar por ciclo.
- Número de recomendaciones dadas.
- Total de problemas resueltos.
- Número de problemas resueltos en el ciclo actual.
- Total de problemas fallados.
- Número de problemas fallados en el ciclo actual.
- Total de incrementos de nivel.

- Número de incrementos de nivel en el ciclo actual.
- Total de alumnos que completaron todos los problemas.
- Total de alumnos que se han rendido.
- Número de veces que el recomendador no pudo dar una recomendación.
- Tiempo transcurrido.

El usuario puede utilizar el sistema para visualizar gráficas de la simulación actual o de las simulaciones realizadas previamente, también se pueden realizar gráficas en las cuales se comparan 2 simulaciones, los tipos de gráficas que maneja el sistema son:

- Root Mean Square Error (sólo SVD).
- Número de recomendaciones dadas.
- Número de problemas fallados.
- Número de problemas fallados por ciclo.
- Número de problemas resueltos.
- Número de problemas resueltos por ciclo.
- Número de incrementos de nivel.
- Número de incrementos de nivel por ciclo.
- Número de usuarios que completaron los problemas.
- Número de usuarios rendidos.
- Número de veces que no se pudo generar recomendación.
- Número de veces utilizado coldStart.
- Número de veces utilizado coldStart por ciclo.
- Precisión de recomendaciones.
- Precisión por recomendaciones.
- Precisión por log de recomendaciones.
- Tiempo tomado para la función “realiza Análisis”.
- Tiempo promedio en dar una recomendación.

El sistema también cuenta con una barra de progreso y un área de texto en la cual se muestra el historial de las recomendaciones y el estado de cada usuario simulado en cada ciclo al terminar la simulación. En la ilustración 11 de muestra la impresión de la pantalla del prototipo.

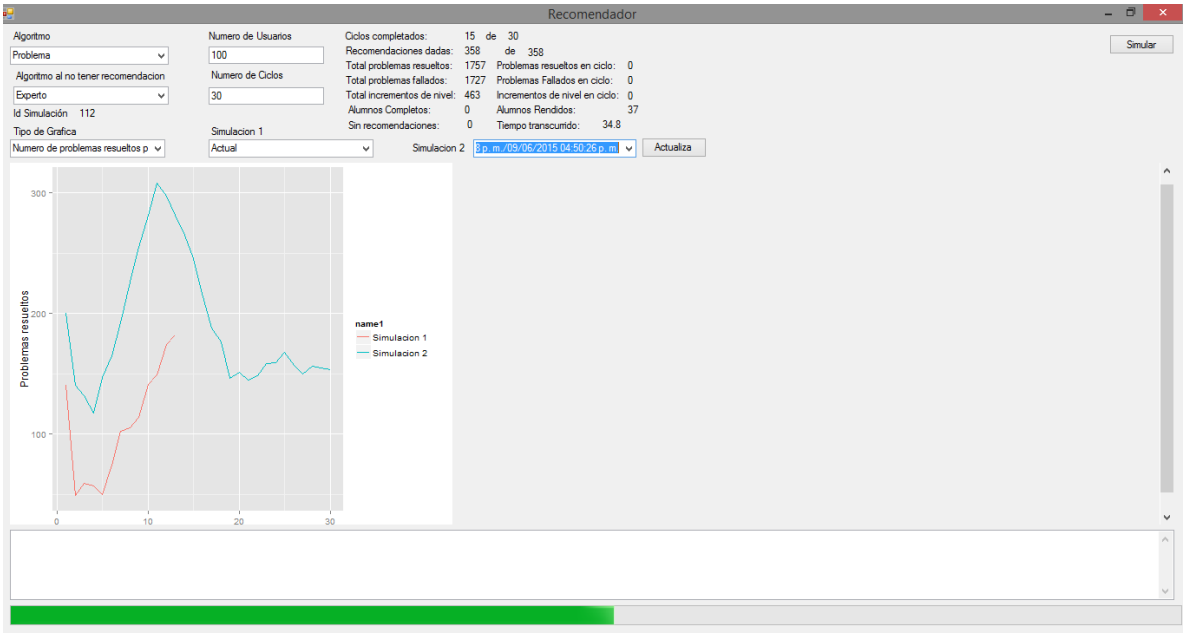


Ilustración 11 Prototipo

Base de Datos

En el siguiente diagrama (Ilustración 12) se pueden visualizar las tablas y sus relaciones de la base de datos utilizada por la simulación. El diagrama entidad relación utiliza la notación de Crow, el diagrama esta coloreado de acuerdo con quien utiliza principalmente la tabla:

- Azul: Tablas en la Base de datos Karelotitlan.
- Verde: Tablas utilizadas por la simulación.
- Naranja: Tabla utilizada por el recomendador basado en un experto.
- Morado: Tabla utilizada por el recomendador basado en inversiones.
- Amarillo: Tabla utilizada por el recomendador basado en similitud de usuarios.
- Rojo: Tabla utilizada por el recomendador basado en similitud de problemas.
- Blanco: Tablas utilizada por el recomendador basado en factorización de matrices.

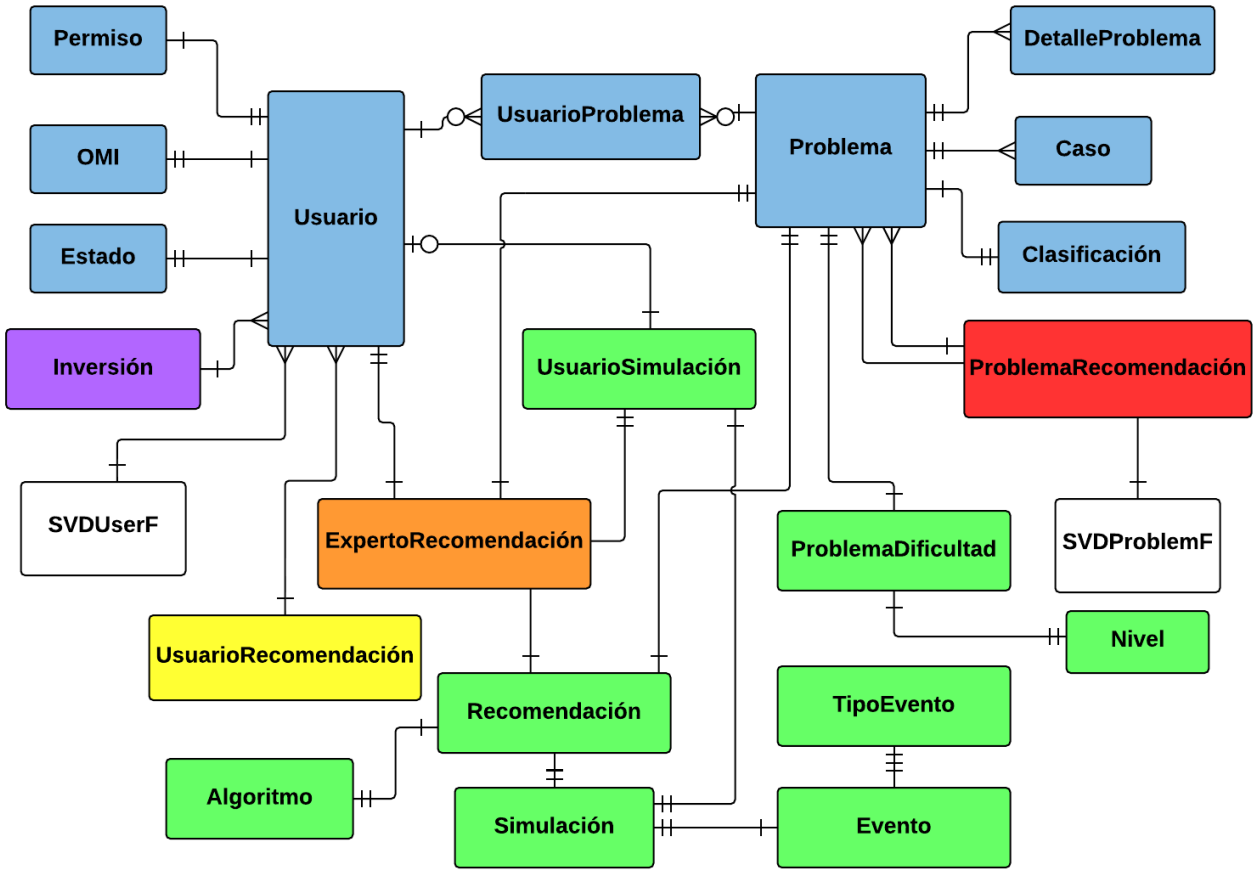


Ilustración 12 Diagrama Entidad Relación de la Base de Datos

Karelotitlán

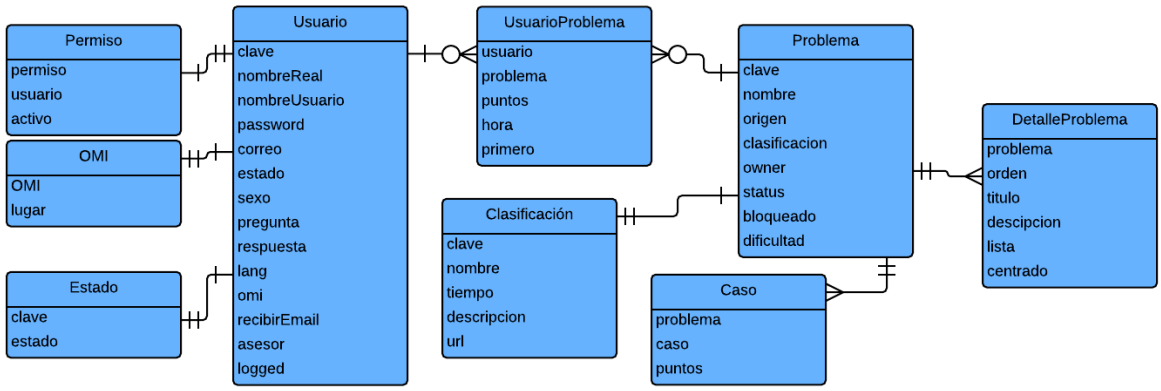


Ilustración 13 Diagrama Entidad Relación Karelotitlán

La base de datos de Karelotitlán (Ilustración 13) mantiene toda la información acerca de los usuarios, problemas y los problemas resueltos por cada usuario.

La tabla principal de esta base de datos es “Usuario Problema” en la que cada registro contiene el identificador de usuario (usuario), identificador de problema (problema), puntos obtenidos en el primer envío (primero), mejor puntuación obtenida hasta el momentos (puntos) y fecha y hora del

mejor envío (hora). Con los registros existentes, esta tabla cuenta con 73,690 registros de problemas intentados desde 2009-01-03 22:01 a 2014-01-26 2:27.

La tabla Usuario Contiene toda la información acerca de los usuarios, estos datos son:

- Clave: Identificador del usuario.
- Nombre Real: nombre completo del Usuario.
- Nombre Usuario: nombre en el sistema del usuario.
- Password: contraseña del Usuario para ingresar al sistema.
- Correo: correo con el cual el usuario se registró.
- Estado: Identificador del estado al que pertenece. Este identificador hace referencia a la Tabla Estado, esa tabla cuenta con 32 registros, DF y Estado de México están considerados como solo un estado y se tiene el Estado -Extranjero- para aquellos usuarios que no son mexicanos.
- Sexo: Identificación del género del usuario.
- Pregunta: pregunta para la recuperación de contraseña.
- Respuesta: Respuesta a la pregunta de recuperación de contraseña.
- Lang: lenguaje de programación preferido.
- Omi: Año de la última olimpiada en la que puede participar el usuario, este campo hace referencia a la tabla OMI la cual contiene 13 registros (de la Olimpiada de 2014 al del 2020), más 6 registros que identifican a los usuarios como Asesor, Líder, Delegado, COMI, Invitado o Comité Estatal.
- Recibir Email: si el usuario desea recibir e-mails.
- Asesor: el identificador del Asesor del Usuario.
- Logged: si el usuario se encuentra conectado.

La base de datos cuenta con 1,711 usuarios registrados.

La tabla Permiso contiene la descripción del acceso que tiene el usuario en la página.

La tabla Problema contiene la información referente a los problemas en la página de Karelotitlán, los datos que contiene son:

- Clave: identificador del problema.
- Nombre: nombre que se le dio al problema.
- origen: quien creo el problema o en qué concurso se publicó.
- Clasificación: Cual es la clasificación del problema, este campo hace referencia a la tabla Clasificación que contiene 6 registros, que son los 6 tipos de problemas que están en la página.
- Owner: Usuario del sistema que publico el problema en la página.
- Status: estado de edición del problema.
- Bloqueado: si por alguna razón este problema no debería ser mostrado en la página.
- Dificultad: Dificultad del problema de acuerdo de la opinión de un experto. Este dato es utilizado para las recomendaciones basadas en la opinión de un experto, su valor es un número entre 1 y 1000, donde 1 es un problema muy sencillo y 1000 es el problema más

difícil. Cabe aclarar que esta clasificación es diferente a la de “Problema Dificultad” que es utilizada para etiquetar el nivel de los usuarios.

En esta tabla se cuenta con 126 problemas.

La tabla “Detalle Problema” tiene la información del problema para el usuario, cada registro es una sección de la descripción del problema, un registro contiene el identificador del problema (problema), el orden de la sección (orden), título de la sección (título), texto a mostrar (descripción), si los párrafos se muestran como una lista (lista) y si el texto debe estar centrado (centrado). Esta tabla cuenta con 584 registros.

La tabla Caso contiene el listado que los casos para cada problema y cuantos puntos se obtiene al resolver ese caso. Esta tabla cuenta con 1,276 registros.

Simulación

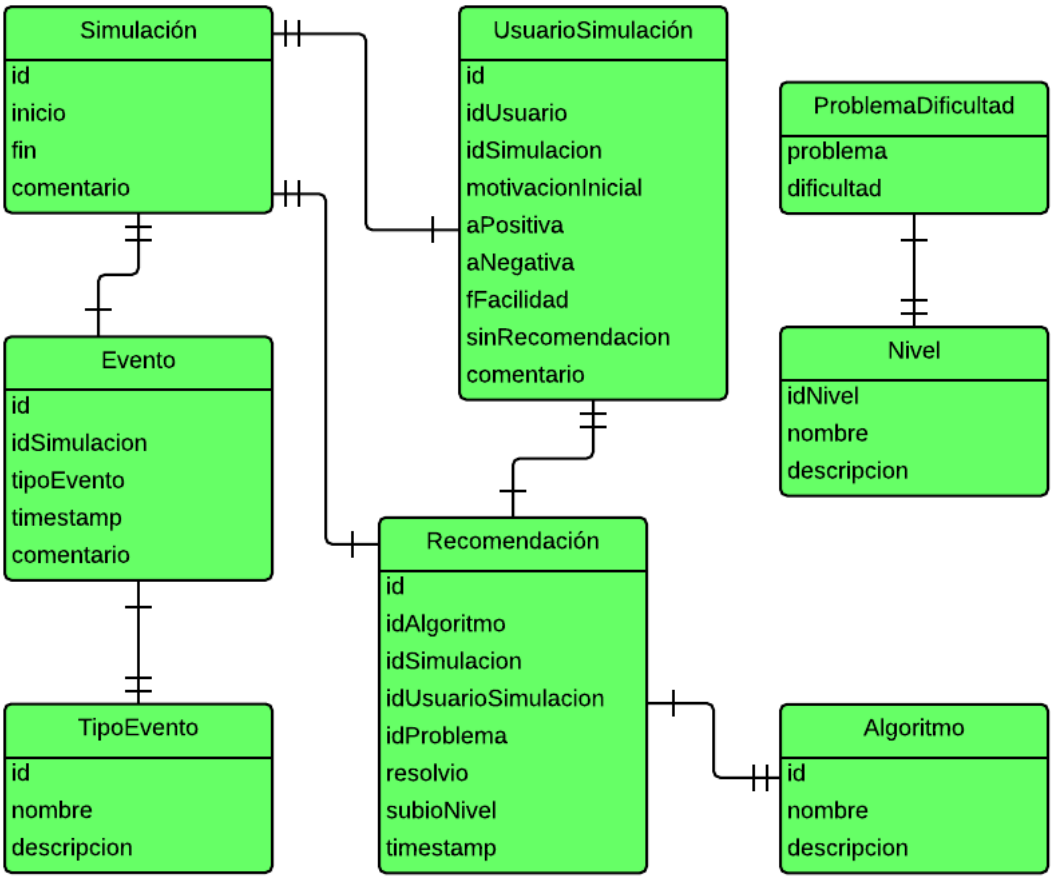


Ilustración 14 Diagrama Entidad Relación Simulación

Un conjunto de tablas son utilizadas por el Simulador, en el diagrama estas tablas se encuentran en color verde.

La tabla Principal de este conjunto es “Recomendación” que registra todas las recomendaciones realizadas por la simulación, cada registro tiene un identificador de registro (id), el identificador de Algoritmo utilizado (idAlgoritmo), el identificador de usuario simulado que se le dio la

recomendación (idUsuarioSimulacion), el identificador del problema que se recomendó (idProblema), si se resolvió el problema (resolvio), si el usuario subió de Nivel (subioNivel) y en qué momento se realizó la recomendación (tiemstamp).

La tabla Algoritmo almacena el nombre de la clase del recomendador utilizado, cada registro contiene un identificador (id), el nombre de la clase (nombre) y la descripción de este recomendador.

La tabla Simulación registra cada una de las simulaciones que se realizó en el sistema, cuando inicio y término.

Para mantener un registro de los usuarios utilizados en cada simulación se creó una tabla intermedia llamada “Usuario Simulación”, para cada usuario ficticio creado en la tabla “Usuario”, se crea un registro en “Usuario Simulación” asignándole un identificador (id), relacionándolo con el Usuario ficticio en Usuario (idUsuario), en que simulación fue creado (idSimulacion), los parámetros utilizados en su simulación: motivación inicial (motivacionInicial), la afectación cuando resuelve correctamente un problema (aPositiva), la afectación cuando no resuelve un problema (aNegativa) y el factor de disminución en caso de existir una gran diferencia en nivel (fFacilidad).

También se prevé la necesidad de registrar diferentes eventos, cada Evento tiene una identificación (id), simulación a la que pertenece (idSimulacion), tipo de evento (tipoEvento), fecha y hora en que ocurrió y comentario.

La tabla ‘Tipo Evento” tiene el registro de todos los posibles eventos que pudieran ocurrir, la información que guarda es el identificador (id), nombre del evento (nombre) y la descripción más detallada de en qué consiste el evento (descripción). Los tipos de evento que se manejan son:

- Root Mean Square Error (RMSE-SVD): Raíz cuadrada del promedio de los errores al cuadrado entre las calificaciones obtenidas y predichas, sólo para el algoritmo SVD. Registrado cada vez que se realiza una iteración de entrenamiento.
- Número de recomendaciones dadas (nRecomendaciones): Número de recomendaciones dadas en cada ciclo de la simulación. Registrado cada fin del ciclo.
- Número de problemas fallados (nFallos): Número total de problemas fallados en lo que se lleva de la simulación. Registrado al final de cada ciclo de simulación.
- Número de problemas fallados por ciclo (nFallosCiclo): Número de problemas fallados en el ciclo. Registrado al final de cada ciclo de simulación.
- Número de problemas resueltos (nExitos): Número de problemas exitosos hasta el momento en la simulación. Registrado al final de cada ciclo de simulación.
- Número de problemas resueltos por ciclo (nExitosCiclo): Número de problemas exitosos por ciclo. Registrado al final de cada ciclo de simulación.
- Número de incrementos de nivel (nIncNivel): Número de incrementos de nivel hasta el momento. Registrado al final de cada ciclo de simulación.
- Número de incrementos de nivel por ciclo (nIncNivelCiclo): Número de incrementos de nivel por ciclo. Registrado al final de cada ciclo de simulación.
- Número de usuarios que completaron los problemas (nCompletos): Número de alumnos completos hasta el momento en la simulación. Registrado al final de cada ciclo de simulación.

- Número de usuarios rendidos (nRendidos): Número de alumnos rendidos hasta el momento. Registrado al final de cada ciclo de simulación.
- Número de veces utilizado coldStart (nColdStart): Número de veces que se necesitó utilizar el recomendador coldstart hasta el momento. Registrado al final de cada ciclo de simulación.
- Número de veces utilizado coldStart por ciclo (nColdStartCiclo): Número de veces que se necesitó utilizar el coldstart en el ciclo. Registrado al final de cada ciclo de simulación.
- Precisión de recomendaciones (precision): Número de problemas aceptados entre número de recomendaciones por ciclo. Registrado al final de cada ciclo de simulación.
- Precisión por log de recomendaciones (precisionRecomendacion2): Precisión por el logaritmo del número de recomendaciones por ciclo. Registrado al final de cada ciclo de simulación.
- Tiempo tomado para función "inicia Recomendador" (tIniciaRecomendador): Tiempo que se toma en realizar inicia Recomendador.
- Tiempo tomado para función "realiza análisis" (tRealizaAnalisis): Tiempo que se toma en realizar realiza análisis. Registrado al final de cada ciclo de simulación.
- Tiempo promedio en dar una recomendación (tPromedioRecomendacion): tiempo promedio que toma en dar una recomendación. Registrado al final de cada ciclo de simulación.

Para poder calcular las probabilidades de resolver un problema y subir de nivel el simulador necesitó saber cuál es el nivel de cada problema para eso se creó la tabla "Problema Dificultad" que contiene el identificador del problema (problema) y el nivel de este problema (dificultad).

La tabla "Nivel: tiene el nivel (idNivel), nombre y descripción de las diferentes dificultades a las cuales se puede clasificar cada problema. Los Niveles que se tiene son:

- -1, Rendido: El usuario ya no sube más problemas.
- 0, Inicia: El usuario inicia con tema.
- 1, Muy facil: Problemas que se utilizan para explicación de tema.
- 2, Facil: Problemas que tiene ligeras modificaciones.
- 3, Medio: Problemas que es necesario pensar para resolverlo.
- 4, Dificil: Problemas que requieren un análisis para ser resueltos.
- 5, Muy Dificil: Problemas más difícil que los niveles anteriores.
- 6, Completo: El usuario completo todos los problemas.

Recomendador en base a un experto

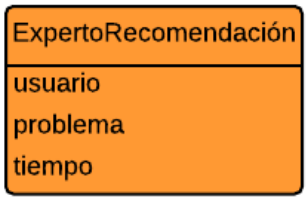


Ilustración 15 Diagrama Tabla ExpertoRecomendación

El recomendador basado en la opinión de un experto adicionalmente de la información que tiene actualmente la página es necesario tener registro que se ha recomendado y hace cuanto se recomendó para que el sistema no se dedique a recomendar lo mismo siempre. Cada registro tiene el id de Usuario al que se le da la recomendación (usuario), que problema se recomendó (problema) y cuando se recomendó (tiempo).

Recomendador basado en Inversiones

Ilustración 16 Diagrama Tabla Inversión

El recomendador en base a las inversiones calcula el número de inversiones entre las historias de los usuarios, cuantos elementos son compartidos, y cuál es el complemento de las historias, esta información es útil para el cálculo del $score$ al momento de dar la recomendación. Esta información es calculada cuando la función "realiza análisis" es llamada, después es guardada en la tabla "Inversión" para ser utilizada al momento de pedir una recomendación.

La tabla "Inversión" (Ilustración 16) cuenta en cada registro con el usuario principal (u1), el usuario con el cual se está comparando (u2), el número de inversiones (inversiones), el número de elementos que comparten las historias de estos usuarios (iguales) y el número de problemas que resolvió u2 pero no u1 (complemento).

Recomendador basado en similitud de usuarios

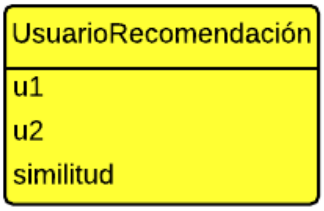


Ilustración 17 Diagrama Tabla UsuarioRecomendación

El recomendador en base a la similitud de usuarios calcula la similitud entre cada par de usuarios después para cada usuario (u1) guarda en la tabla "Usuario Recomendación" los usuarios más similares (u2) y su similitud (similitud), para después ser utilizado durante la recomendación.

Recomendador basado en similitud de problemas

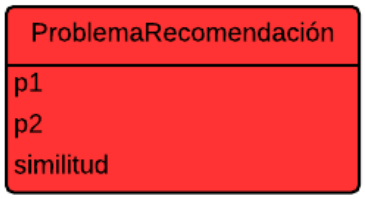


Ilustración 18 Diagrama Tabla ProblemaRecomendación

El modelo de recomendación basado en la similitud de problemas busca la similitud entre cada par de problemas después la similitud es guardada en la tabla "Problema Recomendación", la cual tiene 3 campos: el identificador del primer problema (p1), del segundo problema (p2) y la similitud de estos dos problemas.

Recomendador basado en factorización de matrices

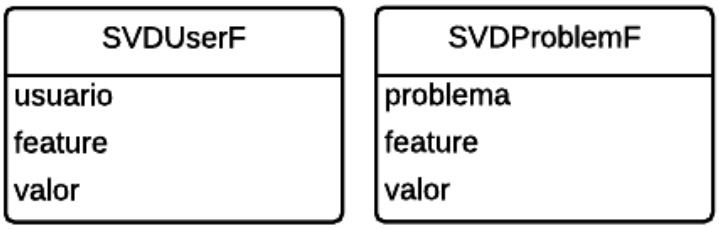


Ilustración 19 Diagrama tablas factorización de matrices

El recomendador basado en factorización de matrices busca encontrar la matriz de factores de los usuarios y de los problemas mientras realiza el análisis, después de esto guarda los valores encontrados en las matrices SVDUserF y SVDProblemF.

SVDUserF guarda para cada usuario (usuario) y para cada característica (feature) el valor de la matriz en esa posición. Mientras que SVDProblmeF para cada problema (problema) y para cada característica (feature) el valor de la matriz.

Clases del sistema

En la Ilustración 20 y 21 se pueden ver los diagramas de clases de los dos componentes más importantes de sistema. Estos dos componentes interactúan por medio de la información en la base de datos de Karelotitlán y son llamados por la interfaz gráfica en dos hilos. Debido a que el proceso de generar una gráfica es tardado la interfaz gráfica envía la petición al generador de gráficas y este responde con un evento cuando la imagen se encuentra lista. El simulador interactúa con los recomendadores gracias a la interfaz "Recomendador", la cual indica que se tienen que implementar las tres funciones básicas de un recomendador.

Simulación

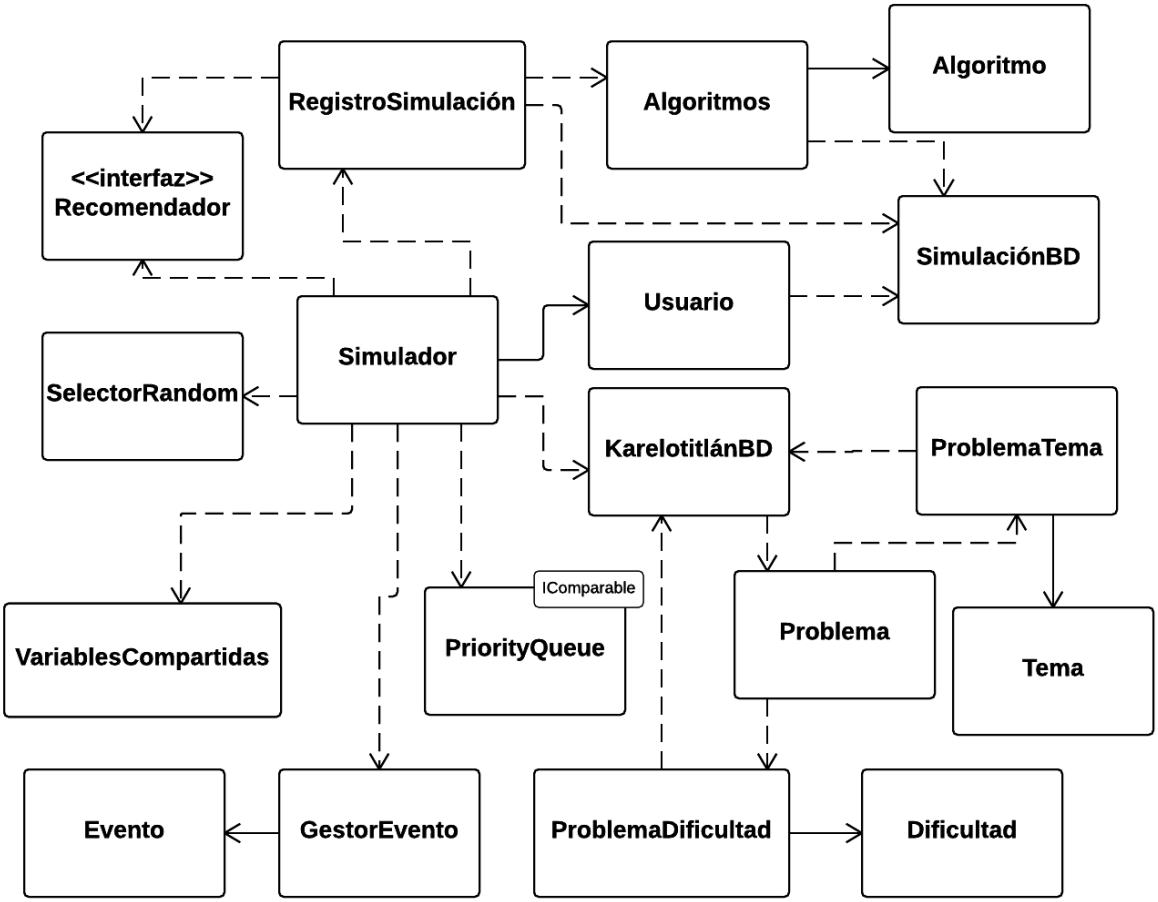


Ilustración 20 Diagrama de clases Simulación

La clase principal de este componente es "Simulador" a partir de esta clase el resto de las clases son accedidas, así como se muestra en la Ilustración 20 las clases que componen este componente son:

- Simulador: Realiza la simulación, tiene diversas variables las cuales son compartidas con la interfaz gráfica para poder estar dando actualizaciones del progreso del algoritmo, y es el encargada de calcular las probabilidades utilizadas para la simulación.
- Recomendador: Es una interfaz que describe que debe de implementar cada uno de los recomendadores.
- Algoritmo: Guarda la información del Modelo de recomendación.
- Problema: Guarda la información de un problema y accede a la información detallada del tema y la dificultad.
- Usuario: Guarda la información del usuario como motivación, habilidades, identificador de simulación e identificador en Karelotitlán, es la clase encargada de realizar el registro de los datos de los usuarios ficticios cuando se crea. Realiza los cambios en motivación y habilidades.
- Tema: Guarda la información referente al tema.
- Dificultad: Guarda la información referente al nivel del problema.
- Evento: Guarda la información referente a un tipo de evento.

- Registro simulación: Encargada de registrar la simulación, obtener sus datos e identificarlos con el resto de las clases que registraran los diferentes eventos y usuarios, registra inicio, fin y algoritmo de la simulación.
- Algoritmos: Encargada de gestionar la identificación y persistencia de los diferentes algoritmos.
- Problema dificultad: Gestiona los detalles de las diferentes dificultades de tal manera que la clase Problema pueda tener los detalles de las dificultades.
- Problema tema: Gestiona los detalles de los diferentes temas.
- Gestor evento: Es la encargada del registro de los eventos.
- Variables compartidas: Gestiona variables compartidas entre la simulación y los modelos de recomendación.
- Selector random: Estructura de datos diseñada especialmente para seleccionar a un elemento al azar con una distribución uniforme.
- Priority Queue: Estructura de datos diseñada para regresar siempre el elemento más valioso de una lista, permitiendo la eliminación del elemento más valioso.
- Karelotitlán BD: Única clase que se le permite el acceso a la base de datos Karelotitlán.
- Simulación BD: Única clase que se le permite el acceso a las tablas de Simulación.

Gráficas

El componente Gráfica contiene las clases “Genera Gráfica” y “GráficaBD”, en la Ilustración 21 se puede observar la relación entre estas clases.

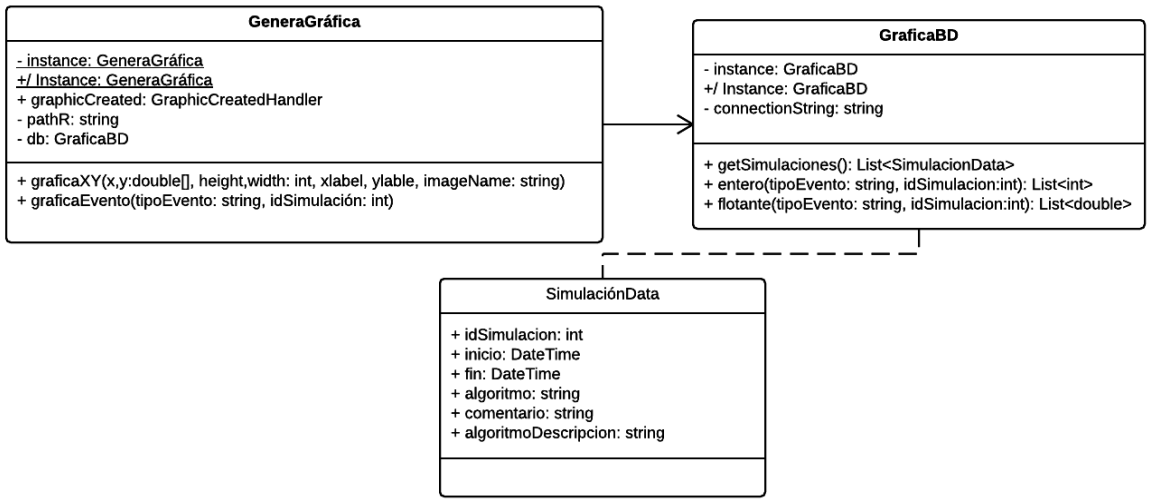


Ilustración 21 Diagrama de clases Gráficas

El objetivo principal de la clase “Genera gráfica” es recibir las peticiones para crear alguna gráfica, acceder a la base de datos con la ayuda de la clase “Gráfica BD”, generar el script de R, correr el script de R y avisar por medio de un evento que el la imagen esta lista.

Por su parte la clase “Gráfica BD” tiene acceso a la tabla de Evento de la cual obtiene la información a ser graficada. Para obtener la información de las diferentes simulaciones utiliza la clase “Simulación Data” que almacena la información de una simulación previa.

Recomendador al azar

Cada uno de los recomendadores hereda e implementa la interfaz “Recomendador” la cual indica las funciones que se deben implementar y son llamadas por el simulador. En la Ilustración 22 se puede observar la clase “RRandom” que es la implementación del recomendador al azar.

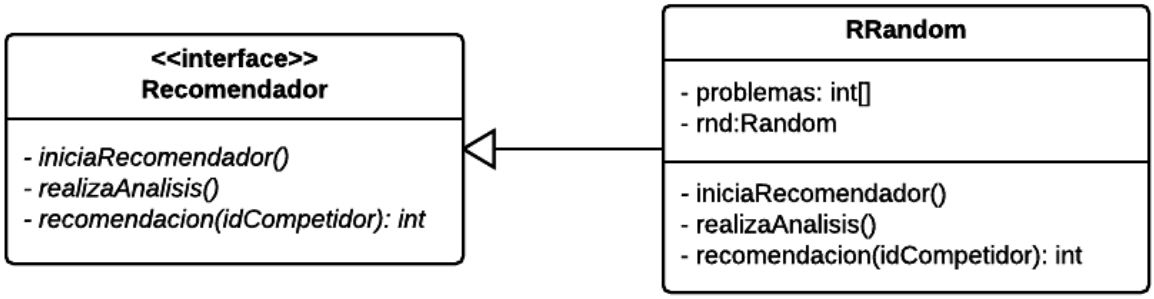


Ilustración 22 Diagrama de clases Recomendador al azar

La clase “RRandom” mantiene la lista de los problemas (problemas) y un generador de números aleatorios (rnd).

Recomendador en base a un experto

En la ilustración 23 se observa las clases que son utilizadas por el recomendador experto. El recomendador Experto (RExperto) utiliza la clase “ExpertoDB” para poder registrar y leer la información requerida por este modelo, es decir, las recomendaciones que se han dado y en que ciclo se dieron esas recomendaciones.

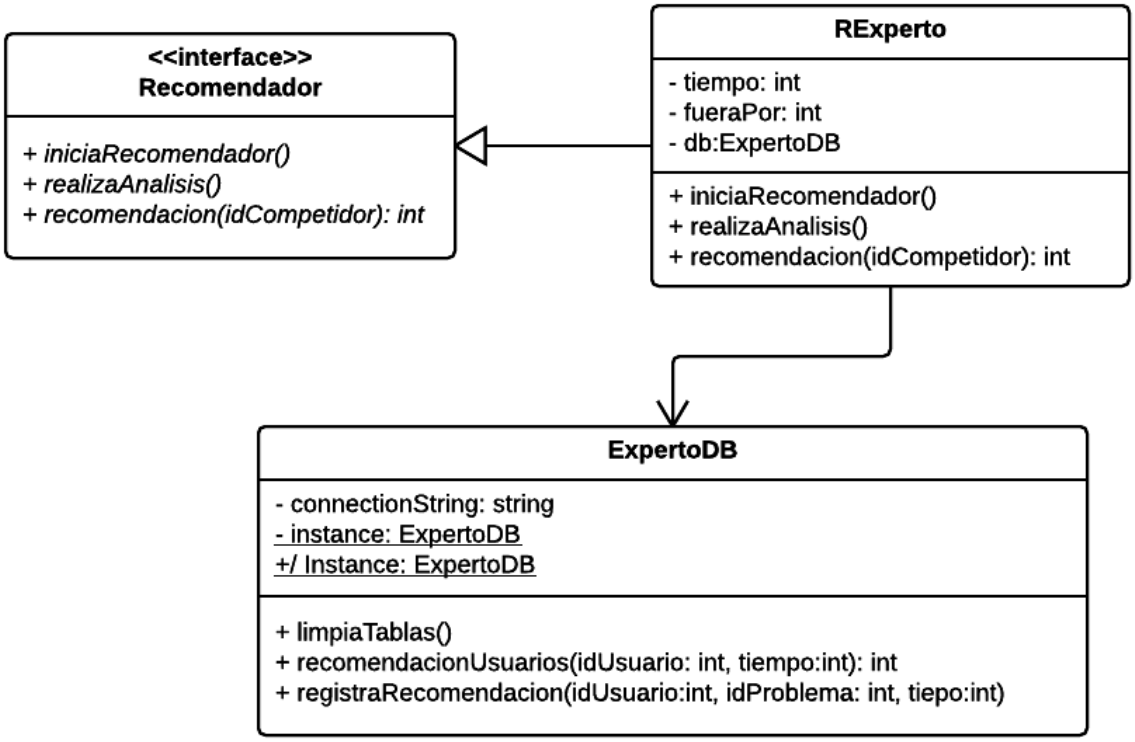


Ilustración 23 Diagrama de clases recomendador en base a un experto

La clase “RExperto” tiene tres miembros:

- Tiempo: El cual identifica en que ciclo se encuentra la simulación.
- Fuera por: Indica por cuantos ciclos no se recomendara de nuevo un problema.
- Db: El acceso a la base de datos.

La clase “Experto DB” controla el acceso a la tabla “Experto Recomendación”, es decir, identifica cual es el siguiente elemento a ser recomendado y registra que problemas ya han sido recomendados.

Recomendador basado en el número de inversiones

En la ilustración 24 se puede observar las clases que se utilizaron para implementar el modelo de inversiones, la clase principal es “RInversión” esta implementa el modelo de recomendación, es apoyada por “InversiónDB” para el acceso a la base de datos y por “Variables compartidas” con la cual comunica con la simulación diferentes variables como cuantas veces ha utilizado “coldStart”.

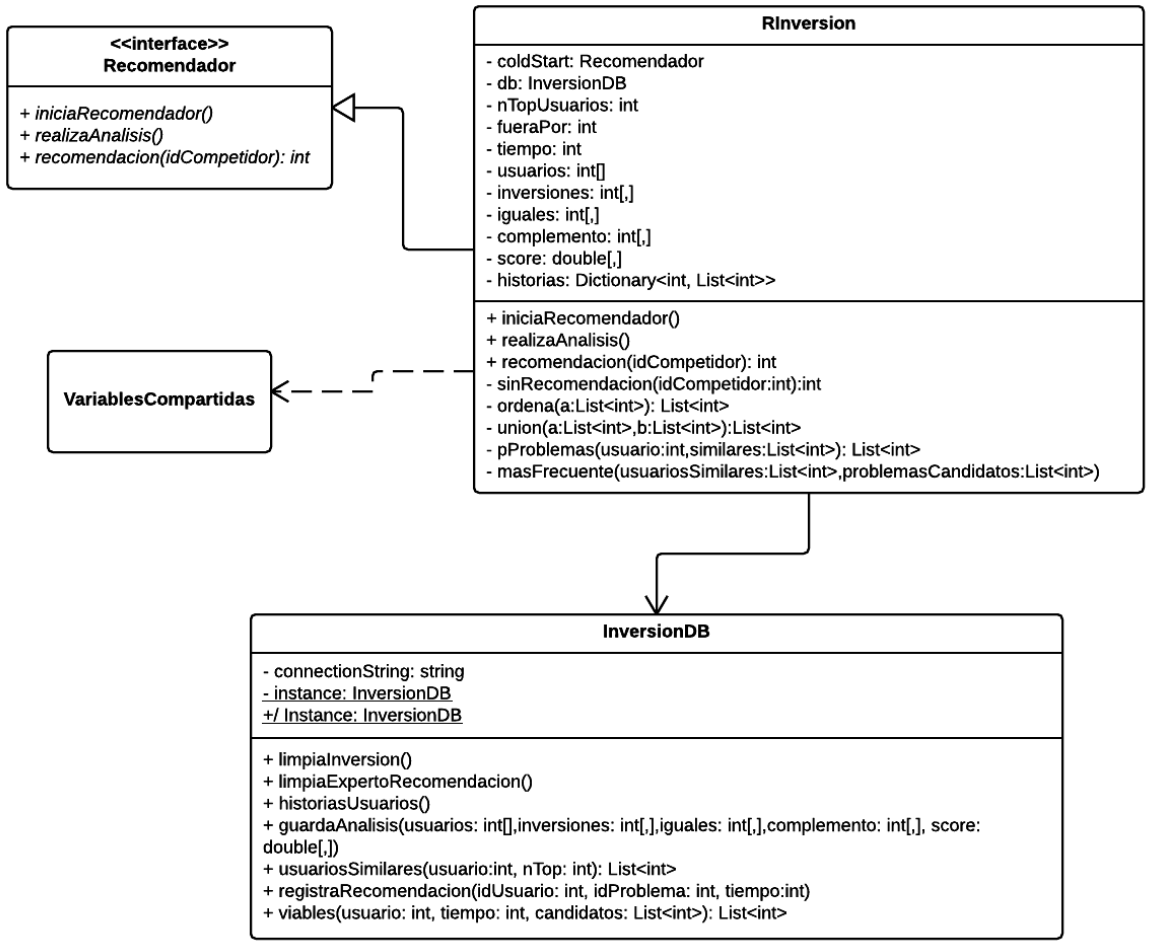


Ilustración 24 Diagrama de clases recomendador basado en inversiones

Inversiones puede ser modificada por: el número máximo de usuarios a ser considerados (nTopUsuarios), que recomendador utilizar cuando no se tiene suficiente información (coldStart), número de ciclos que un problema no se vuelve a recomendar (fueroPor). También esta clase cuenta

con la información de los usuarios, problemas, historias y calcula el número de inversiones, problemas compartidos, complemento y score.

Recomendador basado en similitud de usuarios

El recomendador basado en similitudes de usuarios puede ser modificado por: que recomendador utilizar cuando no se tiene suficiente información (coldStart), número de ciclos que no se considerará un problema una vez que sea recomendado (fueroPor), el número de usuarios que se consideran como mínimo para considerar valida una predicción (minUsuarios), la similitud mínima que deben tener dos usuarios para poderse considerar similar (minSimilitud), el máximo número de usuarios a ser considerados para la predicción (topN).

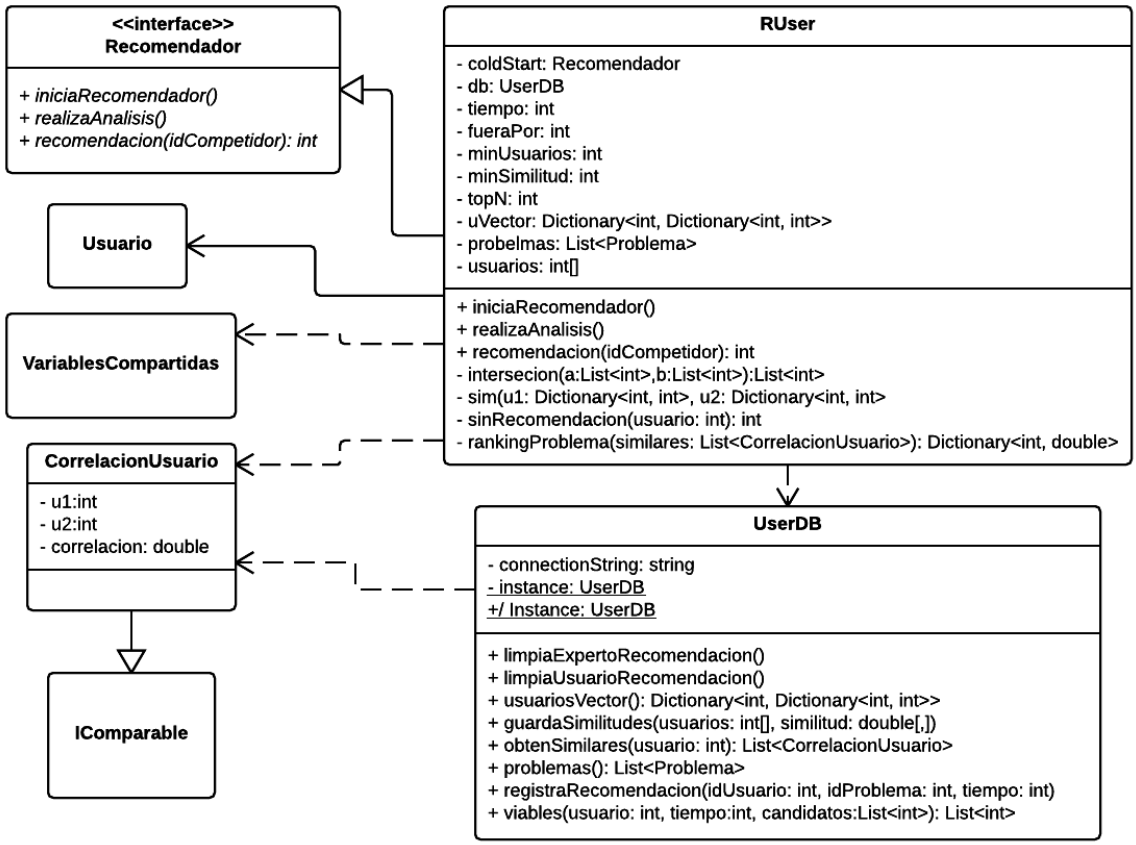


Ilustración 25 Diagrama de clases recomendador basado en similitud de usuarios

La clase “RUser” se apoya de las clases “Usuario”, “variables compartidas”, “UserDB”, “correlación usuario”. “User DB” maneja todos los accesos a la base de datos, correlación junto con la estructura “Priority queue”, ayudan a determinar los usuarios más similares para cada usuario.

Recomendador basado en la similitud de problemas

El recomendador basado en la similitud de problemas hace uso de la clase “Problema”, “variables compartidas” y “Problema BD” como se muestra en la ilustración 26. Este recomendador es afectado por: que recomendador utilizar cuando no se tiene suficiente información (coldStart), número de ciclos que no se considerará un problema una vez que sea recomendado (fueroPor), el mínimo número de problemas necesarios para considerar valida una predicción

(minProblemasIntentados), la similitud mínima que deben tener dos problemas para poderse considerar similar (minimSimilitud), el máximo número de usuarios a ser considerados para la predicción (minProblemas).

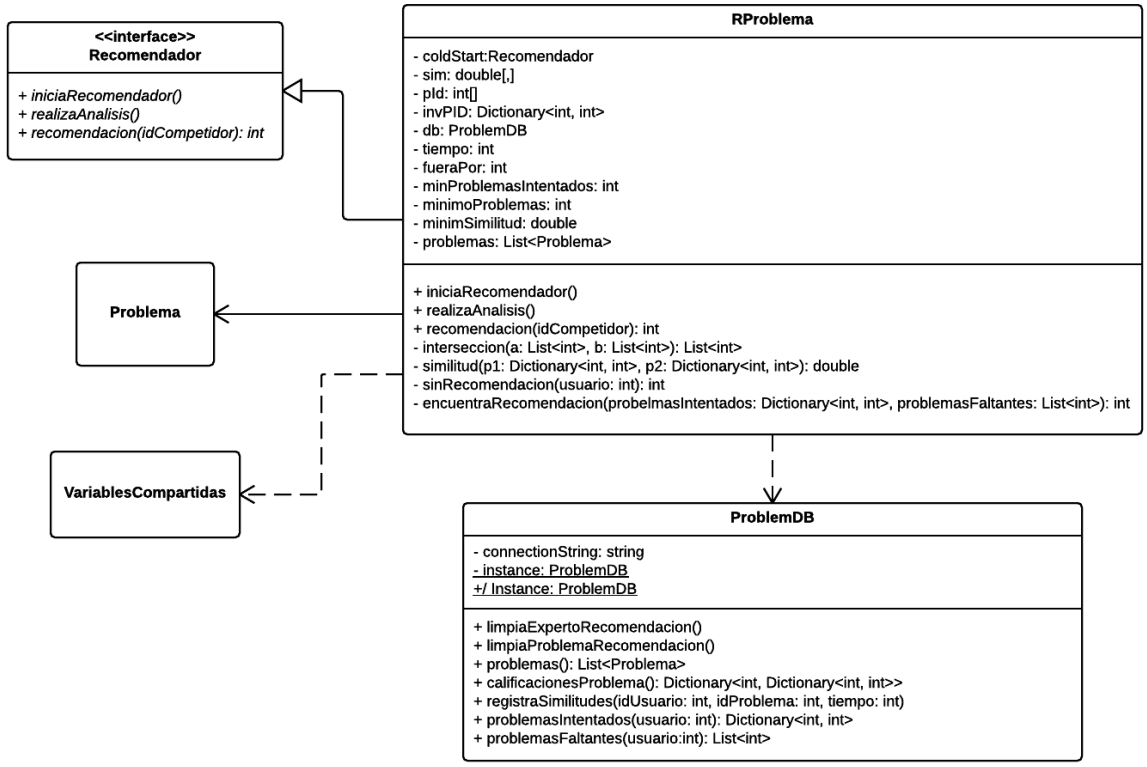


Ilustración 26 Diagrama de clases recomendador basado en similitud de problemas

Recomendador basado en la factorización de matrices

Como se muestra en la ilustración 27 el recomendador basado en la factorización de matrices solo necesita del acceso a la base de datos y a las variables compartidas. Se puede modificar el comportamiento del recomendador por medio de las variables: que recomendador utilizar cuando no se tiene suficiente información (coldStart), número de ciclos que no se considerará un problema una vez que sea recomendado (fueroPor), la razón de aprendizaje (lrate), el número de características a considerar (nFeatures) y el número máximo de iteraciones de aprendizaje (nIterations).

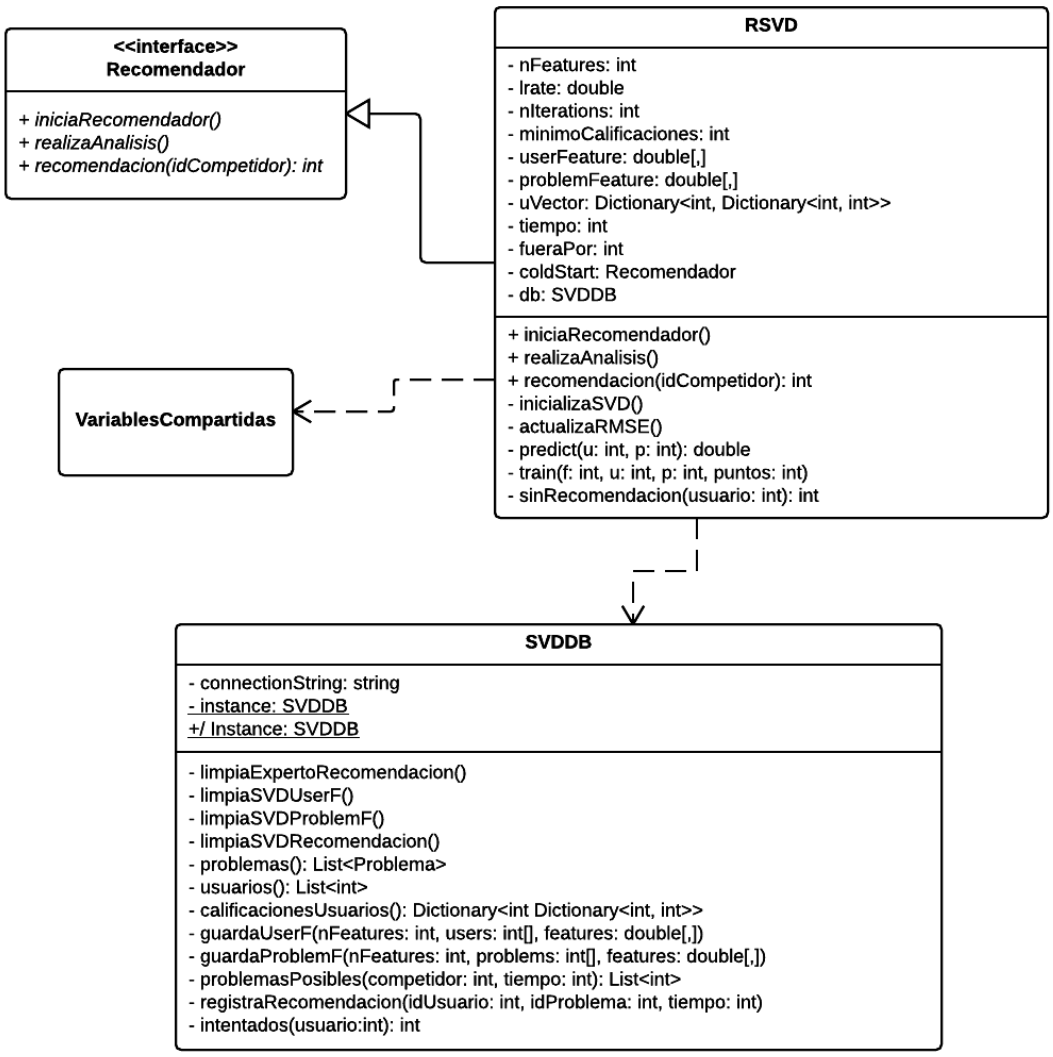


Ilustración 27 Diagrama de clases recomendador basado en factorización de matrices

Capítulo 5: Pruebas al modelo de recomendación

Parámetros de las pruebas

Se probaron los 6 modelos con la información que se tienen de Karelotitlán, esta base de datos tiene las siguientes características:

- Usuarios registrados: 14,960
- Problemas registrados: 126
- Registros desde 2009-01-03 22:01 a 2014-01-26 2:27
- Número de envíos registrados: 73,690
- Información de cada envío: idUsuario, idProblema, Puntuación, Fecha de Envío, puntuación primer envío.

En la ilustración 28 se muestra una gráfica con la frecuencia de usuarios que el número total de envíos se encuentren en alguno de los rangos. Como se puede observar la gran mayoría solo ha resuelto entre 1 a 12 problemas y muy pocos han resuelto de entre 117- a 129 problemas.

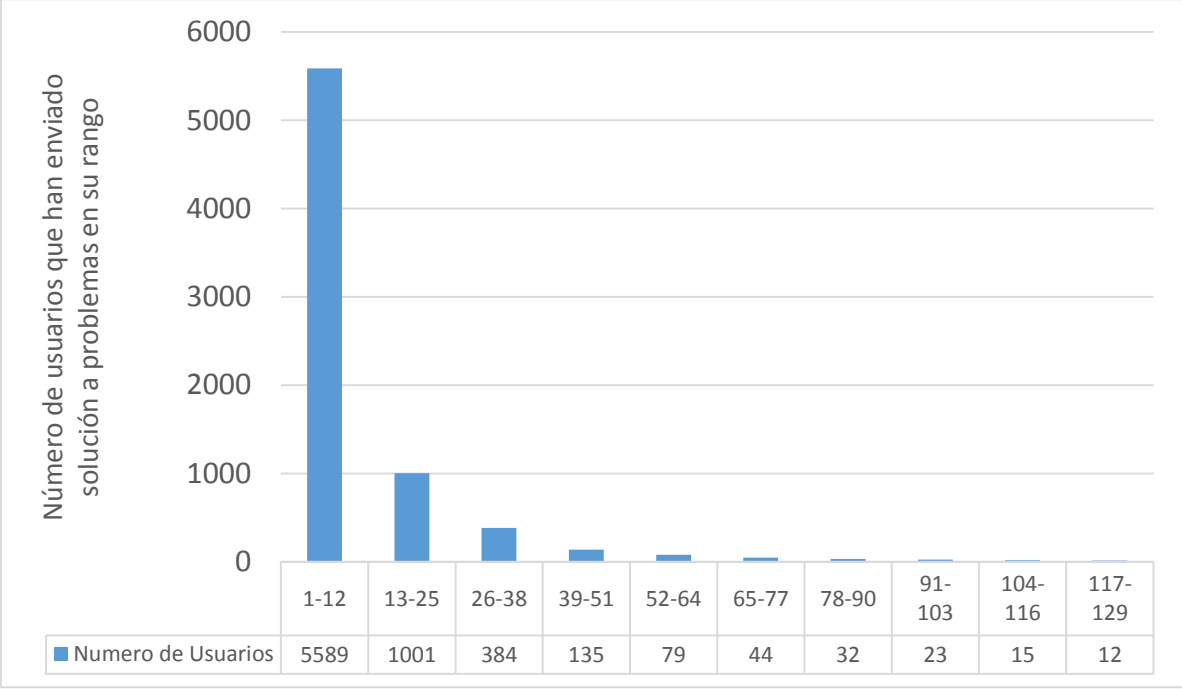
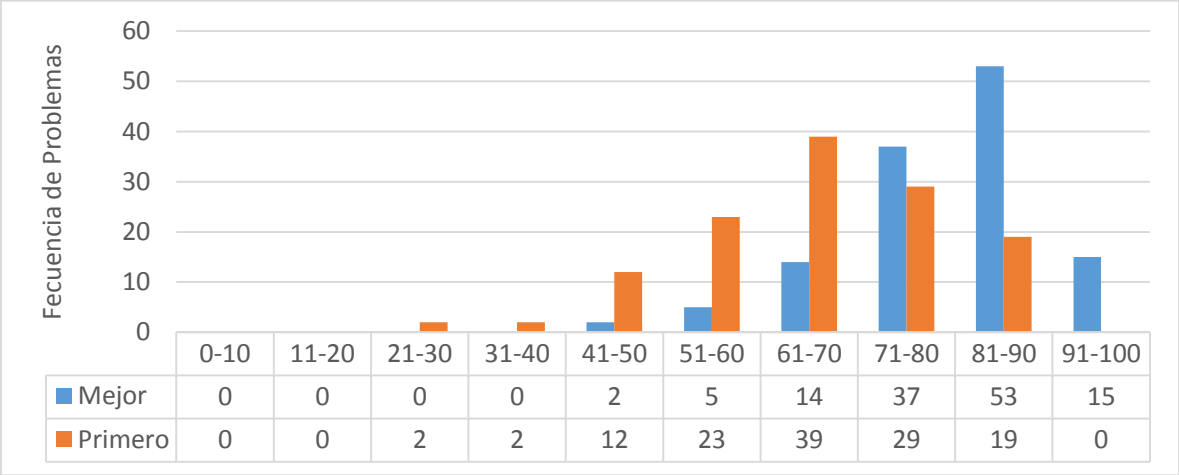


Ilustración 28 Frecuencia de envíos por usuario

Para la ilustración 29 se muestra una gráfica con la frecuencia de cada uno de los promedios de las calificaciones obtenidas por cada uno de los problemas, existen dos calificaciones: la calificación obtenida en el primer envío y la máxima calificación obtenida por el usuario en algún envío

(mejor), como se puede observar la gran mayoría de los problemas tienen un promedio en el rango de 61 a 70 en el primer envío y de 81 a 90 en el mejor envío.



	0-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
Mejor	0	0	0	0	2	5	14	37	53	15
Primero	0	0	2	2	12	23	39	29	19	0

Ilustración 29 Frecuencia del promedio de los problemas

Para cada uno de los modelos de recomendación se inicializaron sus parámetros con los siguientes valores:

- Parámetros de la simulación:
  - Número de ciclos: 30
  - Número de usuarios ficticios: 100
  - Usuarios:
    - Motivación inicial: 2
    - fFacilidad: 1.25
    - aPositiva: 0.5
    - aNegativa: 0.25
- Parámetros de Recomendación al azar
  - Semilla: 123456
- Parámetros recomendación en base a un experto:
  - t_{fuera}: 10
- Parámetros recomendaciones basadas en el número de inversiones:
  - t_{fuera}: 10
  - topN: 5
  - Mínimo de Usuarios: 1
  - coldStart: Recomendación al azar
- Parámetros recomendaciones basadas en similitud de Usuarios
  - t_{fuera}: 10
  - coldStart: Recomendación al azar
  - topN: 20
  - Mínimo de Usuarios: 1
  - Similitud mínima: 0.5
- Parámetros de recomendaciones basadas en similitud de problemas

- t_{fuera}: 10
- coldStart: Recomendación al azar
- Mínimo de Problemas: 3
- Similitud mínima: 0.5

- Parámetros de recomendaciones basadas en factorización de matrices
  - t_{fuera}: 10
  - coldStart: Recomendación al azar
  - Número de características: 16
  - Taza de aprendizaje: 0.001
  - Diferencia para considerar convergencia: 0.01
  - Máximo número de iteraciones: 20

Resultados de la ejecución de las pruebas

Durante las simulaciones se registraron diferentes eventos, en la Tabla 5 se muestran el resumen de estas mediciones. En las ilustraciones 30 a la 37 se muestran las gráficas de barras de estos datos.

Algoritmo	Recomendaciones	Éxitos	Fallos	Incrementos	Completos	Rendidos	ColdStart
Azar	3,427	1,428	1,999	388	0	70	0
Experto	10,460	5,675	4,785	1,156	2	18	0
Inversiones	15,040	8,761	6,279	1,906	0	14	1,210
Usuarios	15,040	8,947	6,350	1,869	0	2	541
Problemas	9,300	5,152	4,143	1,275	0	29	353
SVD	16,650	9,348	6,887	2,015	0	0	0

Tabla 5 Resultados simulación de modelos

El número de recomendaciones otorgadas es el total de veces los usuarios pidieron una recomendación, cada ciclo los usuarios piden recomendaciones de acuerdo a su motivación, un alto número de recomendaciones indica que se mantiene por un largo periodo a los alumno y que estos tienen una motivación alta. En la Ilustración 30 se muestran la gráfica de barras del total de recomendaciones que dieron cada uno de los modelos de recomendación.

Ilustración 30 Gráfica número de recomendaciones dadas

El número de éxitos es el total de problemas que se resolvieron, un alto número de éxitos indica que los usuarios constantemente resolvían los problemas recomendados. En la Ilustración 31 se muestran la gráfica de barras de los éxitos obtenidos por cada uno de los modelos.

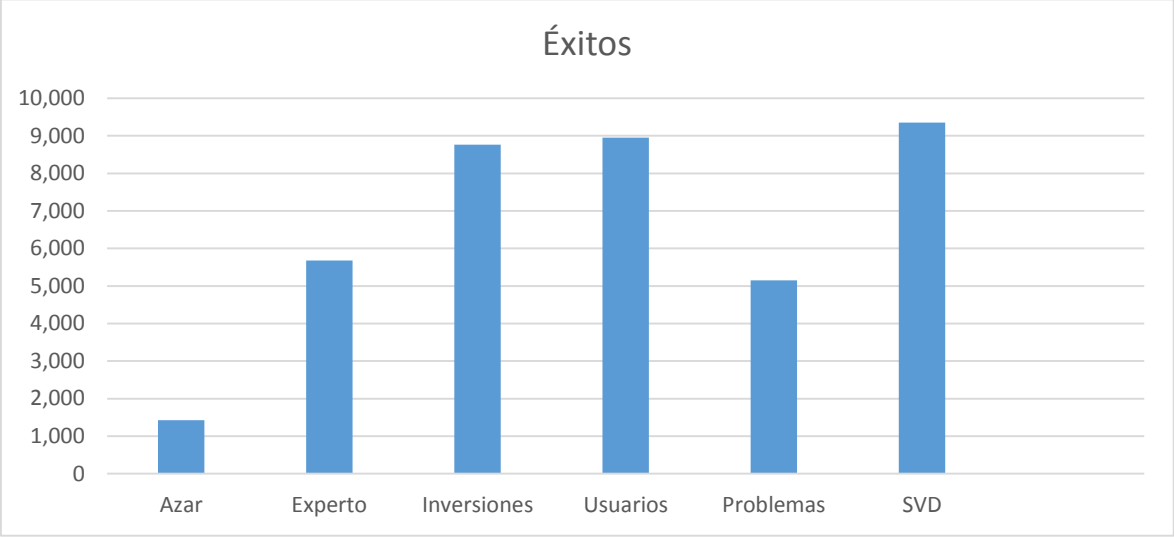


Ilustración 31 Gráfica número de problemas resueltos

Por su parte también se puede considerar los problemas que no se pudieron resolver, esto indica que tan seguido los usuarios no pudieron resolver los problemas de las recomendaciones dadas, debido a que está relacionado con el número de recomendaciones puede ser un poco complicado utilizar esta variable para determinar cuál modelo es mejor, más adelante en este documento se muestra la precisión (número de éxitos entre recomendaciones). En la ilustración 32 se muestran los fallos de cada modelo en la simulación.

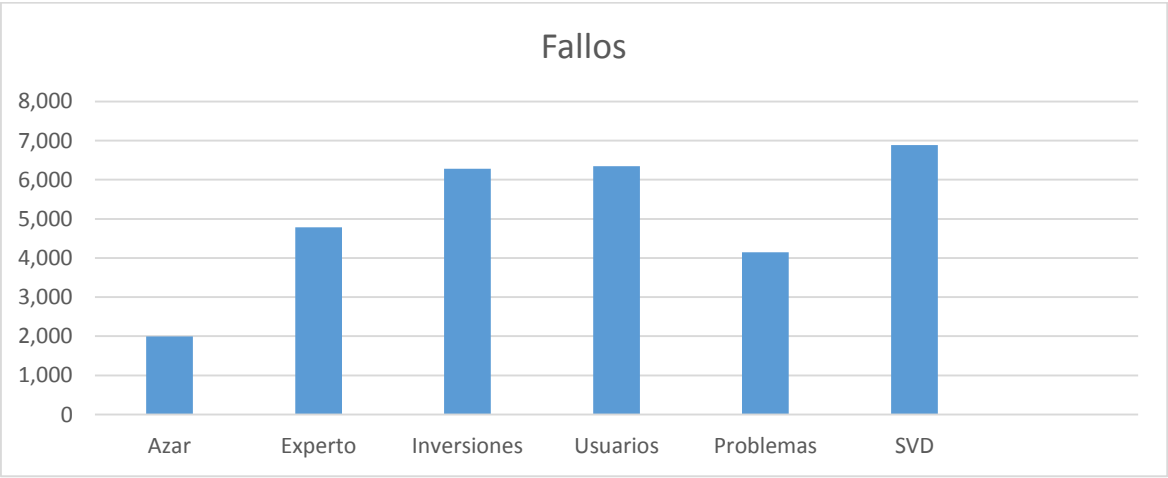


Ilustración 32 Gráfica número de problemas fallados

Un factor importante para ver es el nivel de los usuarios, esto es el número de veces que se decidió subir el nivel de algún usuario, si hay muchos incrementos de nivel indica que el algoritmo ayudo a los usuarios a tener una mayor habilidad. En la ilustración 33 se muestran los resultados obtenidos.

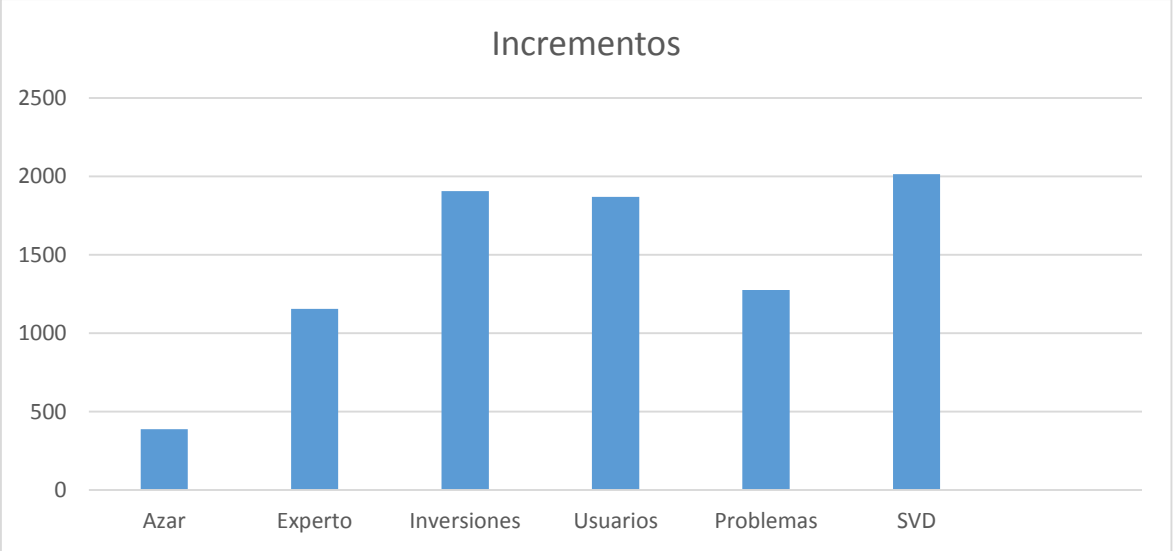


Ilustración 33 Gráfica número de incrementos de nivel

Un usuario que no continua con las recomendaciones es un caso grave, es decir, una vez que un usuario decide desistir de intentar más problemas ya no puede mejorar, por esta razón este parámetro es muy útil para identificar que algoritmos están desmotivando bastante a sus usuarios. En la ilustración 34 se muestra la gráfica de barras del número de usuarios rendidos con cada recomendador.

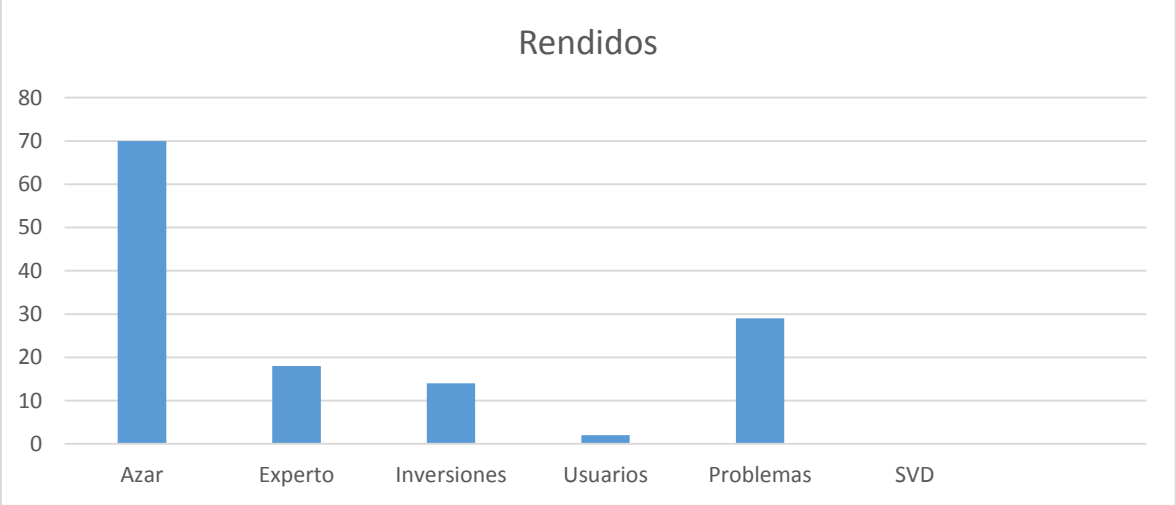


Ilustración 34 Gráfica número de usuarios rendidos

Al contrario de los usuarios rendidos los usuarios que completaron todos los problemas son usuarios que dominan todos los temas y es un parámetro que se buscaría incrementar. En las pruebas realizadas el único recomendador que tuvo usuarios con todos los problemas completos fue el basado en un experto. En la ilustración 35 se muestran el total de usuarios que completaron todos los problemas en cada modelo de recomendación.

Ilustración 35 Gráfica número de usuarios con todos los problemas resueltos

También los recomendadores pueden apoyarse de otro recomendador cuando no se posee suficiente información, de esta manera un recomendador que utiliza mucho la llamada al algoritmo de apoyo indica que constantemente no posee suficiente información. En la ilustración 36 se muestra la gráfica de barras del número de veces que cada recomendador utilizó el recomendador auxiliar.

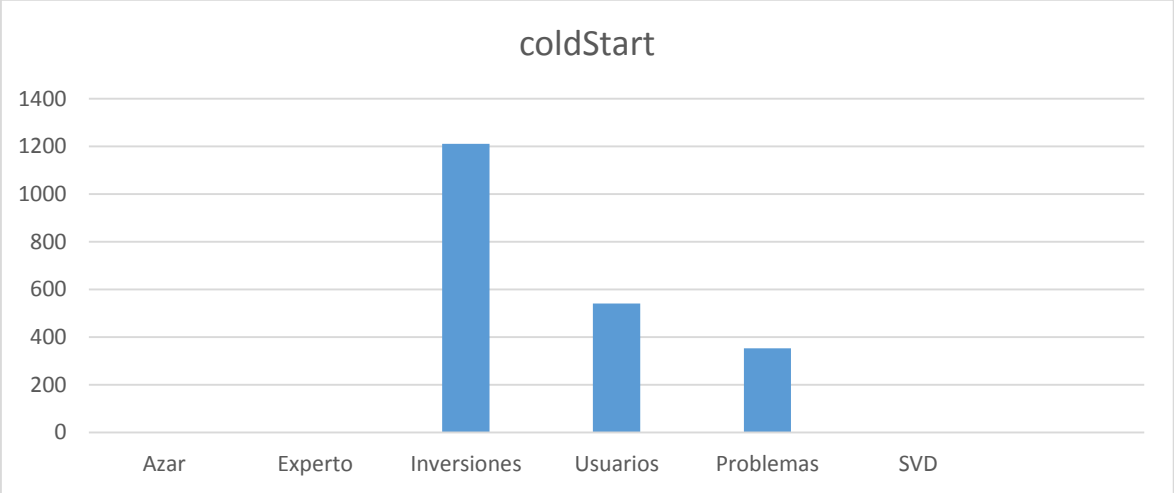


Ilustración 36 Gráfica número de veces que se requirió llamar a coldStart

A continuación se muestran las gráficas generadas por cada una de las simulaciones:

Número de recomendaciones dadas

En las siguientes gráficas (Ilustración 37-42) muestran el número de recomendaciones dadas en cada iteración, otra forma de interpretar este valor es la suma de todas las motivaciones de los usuarios en cada iteración. Es por eso que esta es una medida para poder identificar que recomendadores motivan mejor a los alumnos.

En las gráficas se puede observar las recomendaciones al azar tiende a decrecer la motivación de los alumnos, quienes constantemente tienen crecimiento son las recomendaciones basadas en inversiones, usuarios y factorización de matrices, mientras que las recomendaciones basadas en la similitud de problemas tiene un crecimiento constante pero no tan rápido como inversiones, usuarios o SVD.

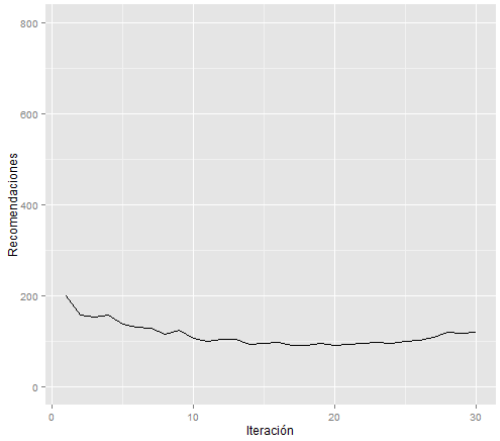


Ilustración 37 Gráfica Número de recomendaciones dadas – Azar

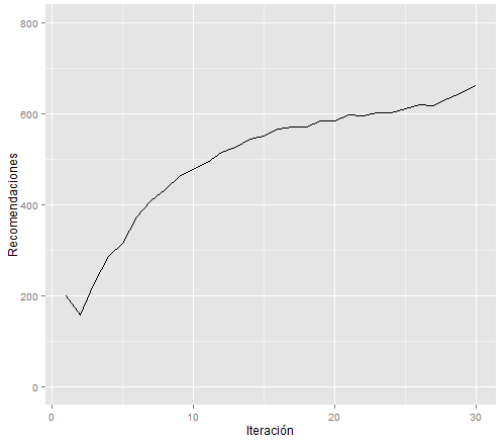


Ilustración 39 Gráfica Número de recomendaciones dadas – Inversiones

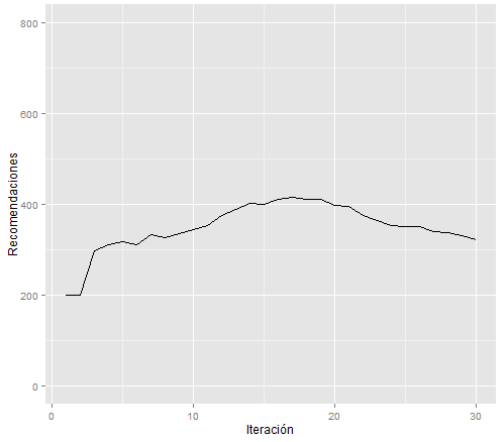


Ilustración 38 Gráfica Número de recomendaciones dadas – Experto

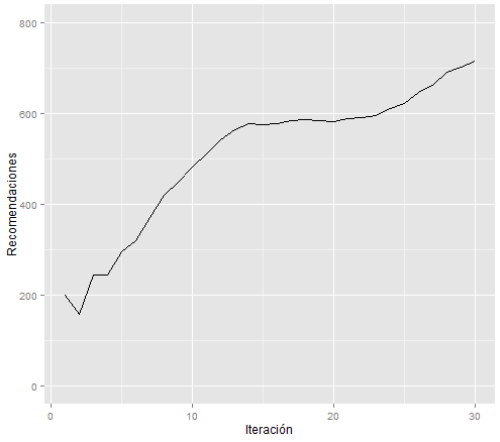


Ilustración 40 Gráfica Número de recomendaciones dadas – Usuarios

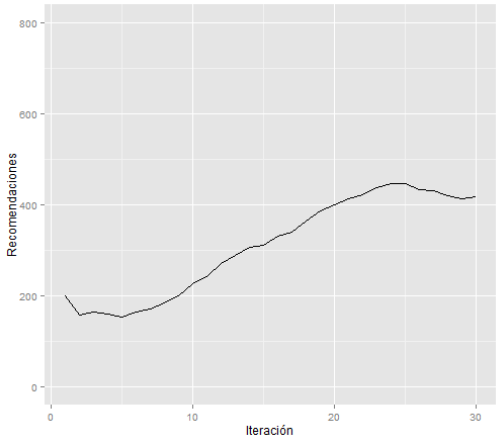


Ilustración 41 Gráfica Número de recomendaciones dadas – Problemas

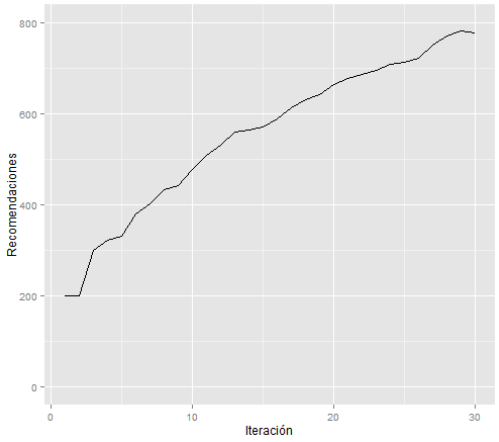


Ilustración 42 Gráfica Número de recomendaciones dadas – SVD

Número de problemas fallados

En las siguientes gráficas (Ilustraciones 43-48) muestran el total de problemas fallados a lo largo de la simulación. Se puede observar que los recomendadores que tienen más fallas son aquellos que tienen mayor cantidad de recomendaciones.

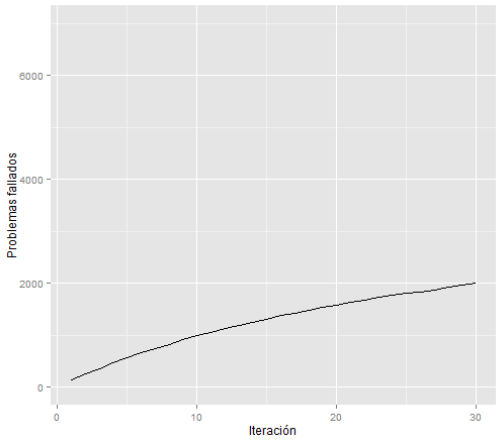


Ilustración 43 Gráfica Número de problemas fallados – Azar

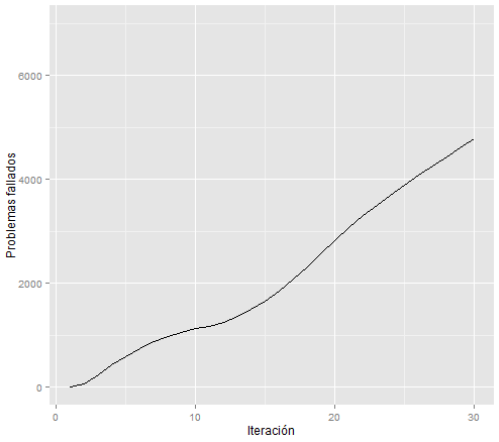


Ilustración 44 Gráfica Número de problemas fallados – Experto

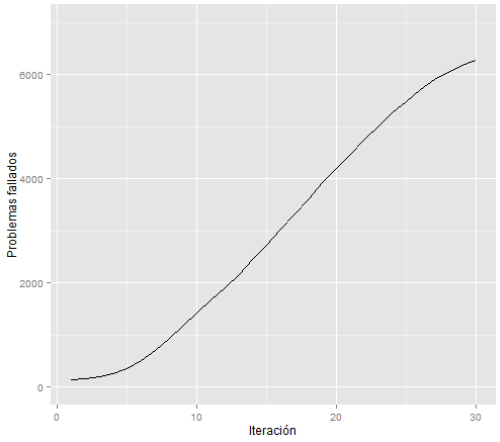


Ilustración 45 Gráfica Número de problemas fallados – Inversiones

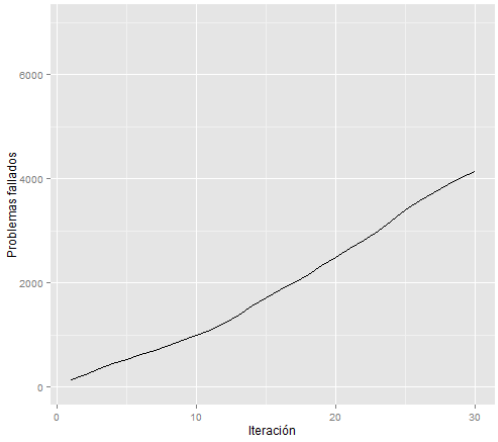


Ilustración 47 Gráfica Número de problemas fallados – Problemas

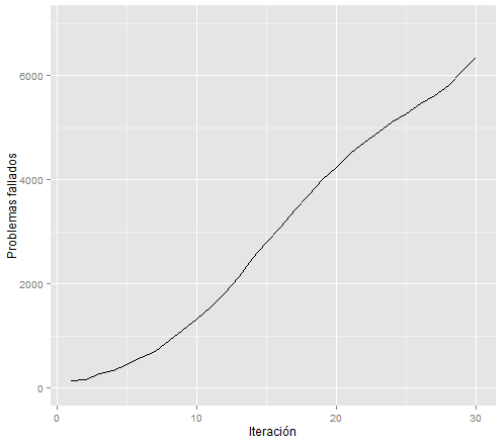


Ilustración 46 Gráfica Número de problemas fallados – Usuarios

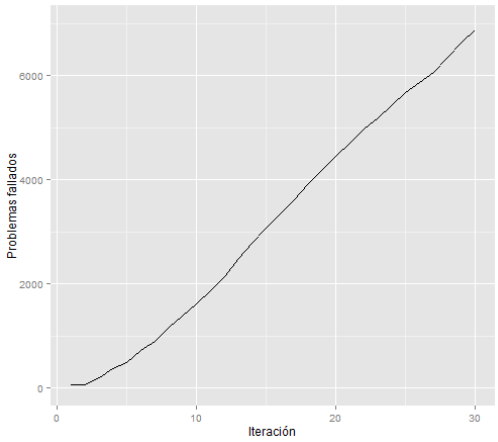


Ilustración 48 Gráfica Número de problemas fallados – SVD

Número de problemas fallados por ciclo

En las ilustraciones 49-54 se muestran las gráficas del número de problemas fallados por ciclo, a diferencia de las gráficas anteriores estas muestran con mayor detalle como evolucionaron los recomendadores, es decir, en que comentos comenzaron a tener menos fallos o más.

Se puede observar que alrededor de la iteración 10 el recomendador experto redujo el número de fallos, el recomendador basado en inversiones rápidamente redujo el número de fallos al inicio al igual que el de factorización de matrices y el basado en similitud de usuarios.

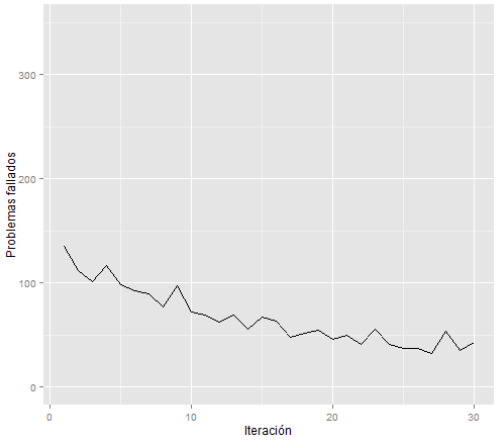


Ilustración 49 Gráfica Número de problemas fallados por ciclo – Azar

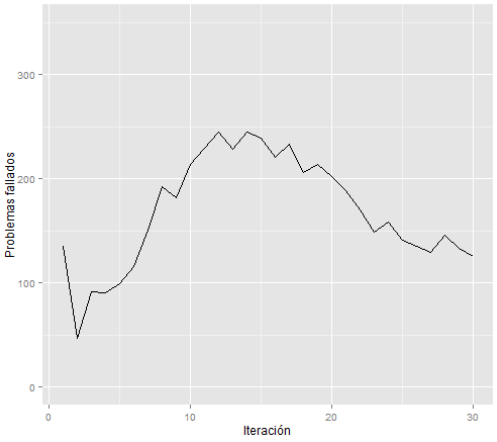


Ilustración 52 Gráfica Número de problemas fallados por ciclo – Usuarios

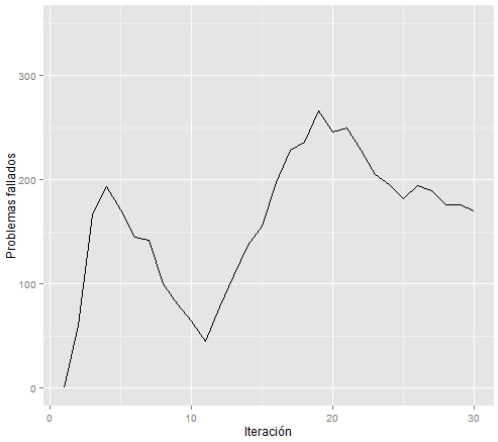
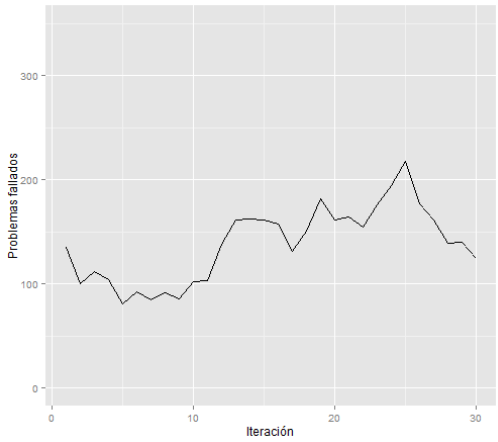


Ilustración 50 Gráfica Número de problemas fallados por ciclo – Experto



.

Ilustración 53 Gráfica Número de problemas fallados por ciclo – Problemas

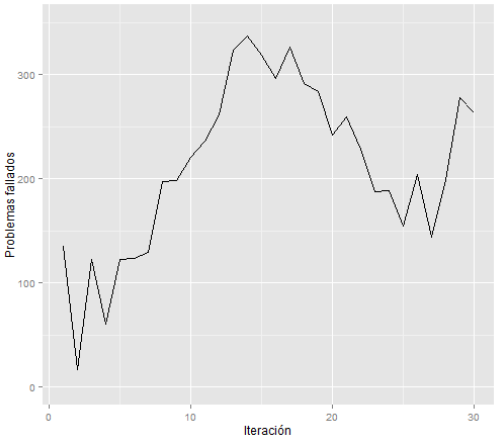


Ilustración 51 Gráfica Número de problemas fallados por ciclo – Inversiones

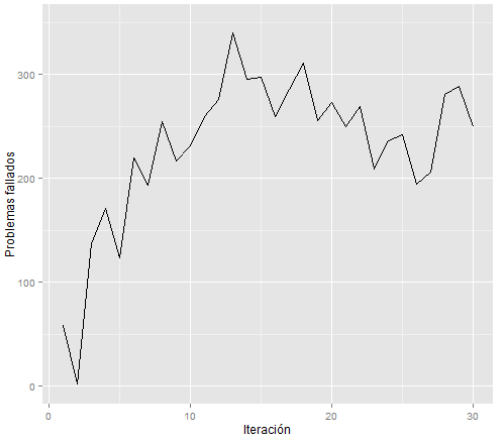


Ilustración 54 Gráfica Número de problemas fallados por ciclo – SVD

Número de problemas resueltos

En las Ilustraciones 55-60 se muestra el total de problemas resueltos, al igual que el número de fallos los recomendadores que presentaron mayor número de recomendaciones también presentaron mayor número de problemas resueltos.

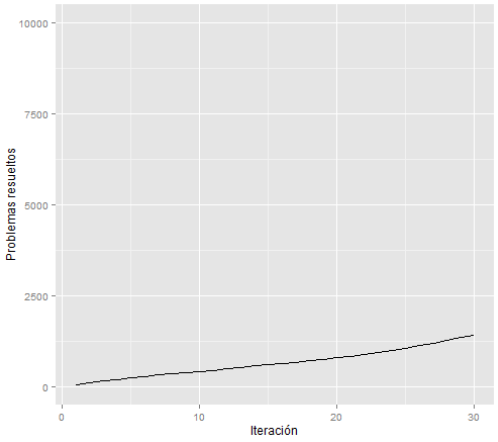


Ilustración 55 Gráfica Número de problemas resueltos – Azar

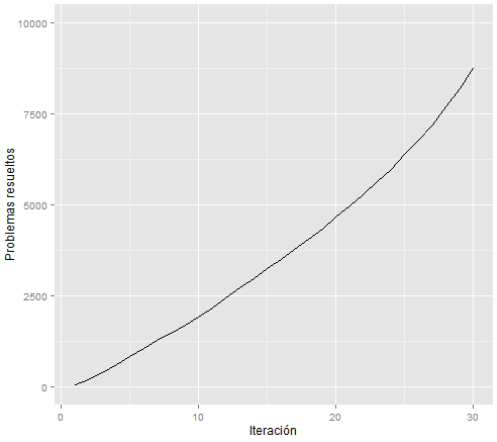


Ilustración 57 Gráfica Número de problemas resueltos – Inversiones

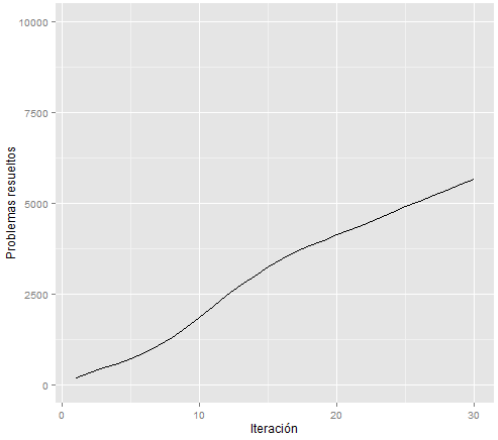


Ilustración 56 Gráfica Número de problemas resueltos – Experto

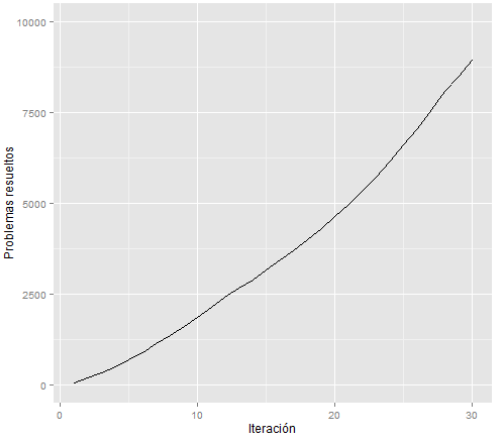


Ilustración 58 Gráfica Número de problemas resueltos – Usuarios

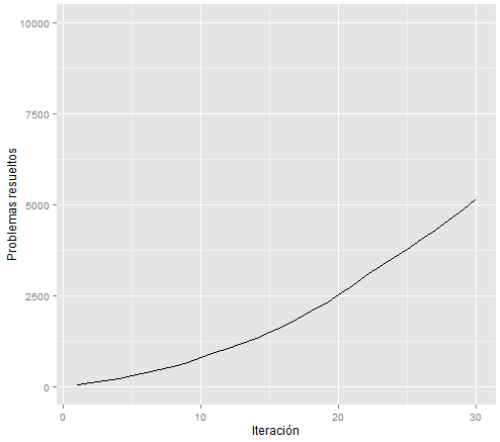


Ilustración 59 Gráfica Número de problemas resueltos – Problemas

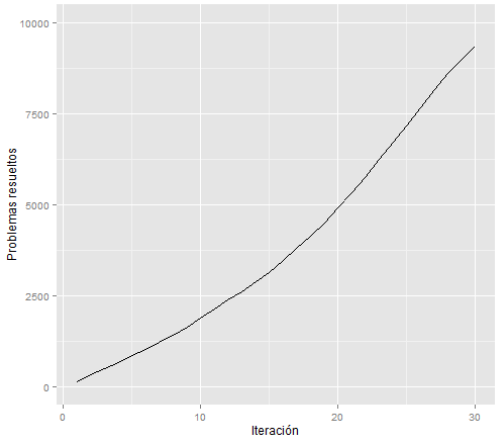


Ilustración 60 Gráfica Número de problemas resueltos – SVD

Número de problemas resueltos por ciclo

En las ilustraciones 61-66 muestran el número de problemas resueltos por ciclo, se puede observar que aquellos recomendadores que desde el inicio lograron superar los 200 problemas resueltos desde el inicio son los recomendadores que obtuvieron mejores resultados.

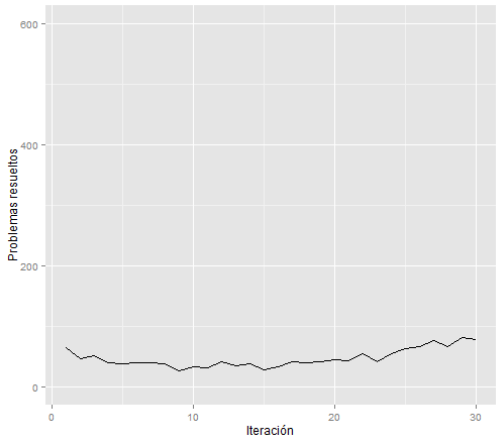


Ilustración 61 Gráfica Número de problemas resueltos por ciclo – Azar

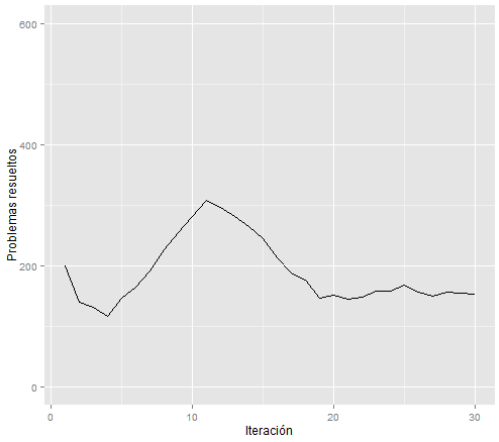


Ilustración 62 Gráfica Número de problemas resueltos por ciclo – Experto

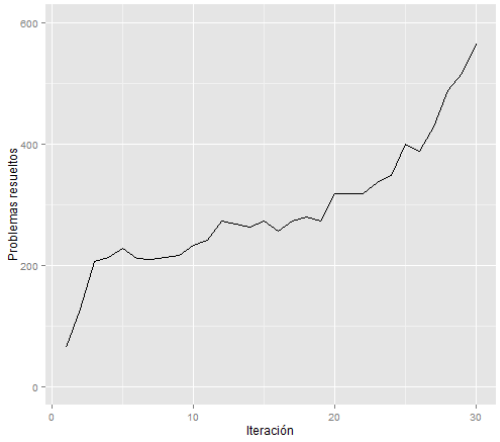


Ilustración 63 Gráfica Número de problemas resueltos por ciclo – Inversiones

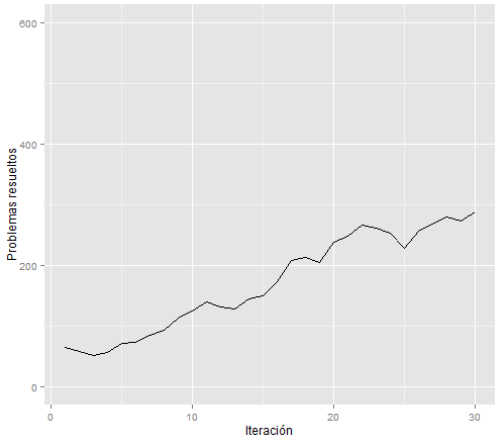


Ilustración 65 Gráfica Número de problemas resueltos por ciclo – Problemas

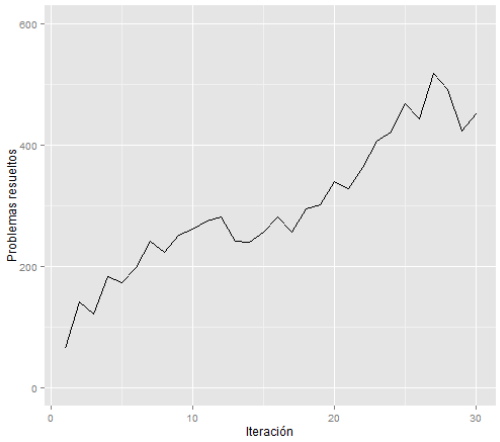


Ilustración 64 Gráfica Número de problemas resueltos por ciclo – Usuarios

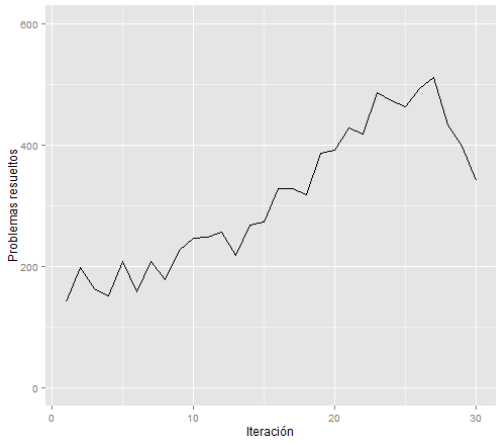


Ilustración 66 Gráfica Número de problemas resueltos por ciclo – SVD

Número de incrementos de nivel

Un parámetro importante para poder identificar el nivel promedio de los usuarios es el número de incrementos de nivel que ocurrieron durante la simulación. De la Ilustración 67 a la 72 se muestran el número de incrementos de nivel durante la simulación de cada uno de los recomendadores.

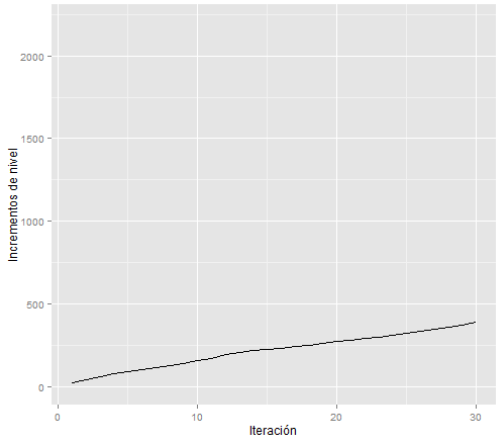


Ilustración 67 Gráfica Número de incrementos de nivel – Azar

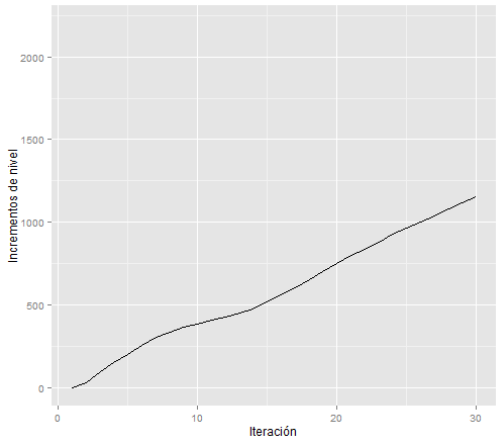


Ilustración 68 Gráfica Número de incrementos de nivel – Experto

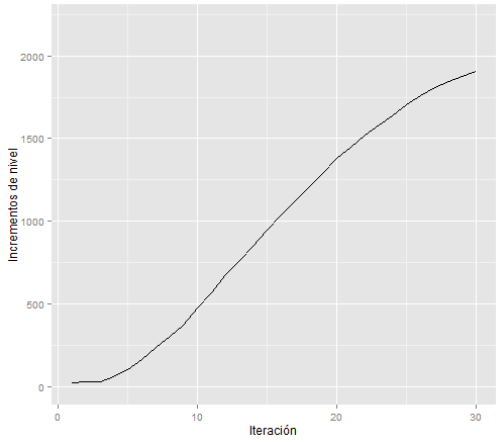


Ilustración 69 Gráfica Número de incrementos de nivel – Inversiones

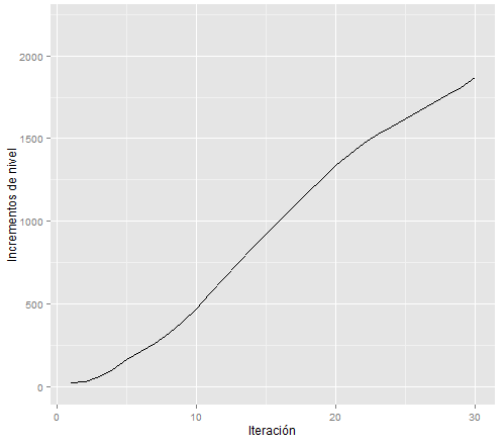


Ilustración 70 Gráfica Número de incrementos de nivel – Usuarios

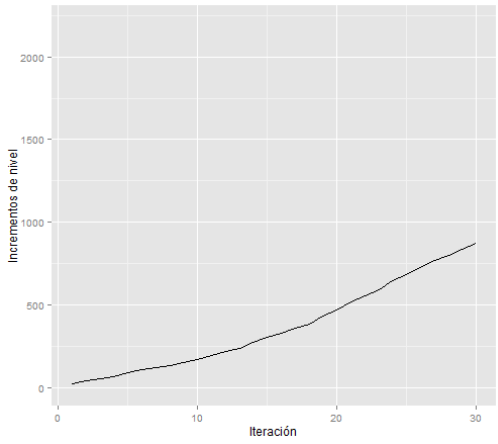


Ilustración 71 Gráfica Número de incrementos de nivel – Problemas

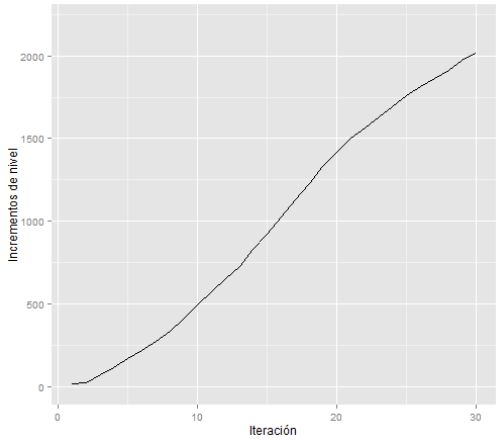


Ilustración 72 Gráfica Número de incrementos de nivel – SVD

Número de incrementos de nivel por ciclo

En las ilustraciones 73-78 se muestran el número de incrementos de nivel en cada ciclo de cada simulación.

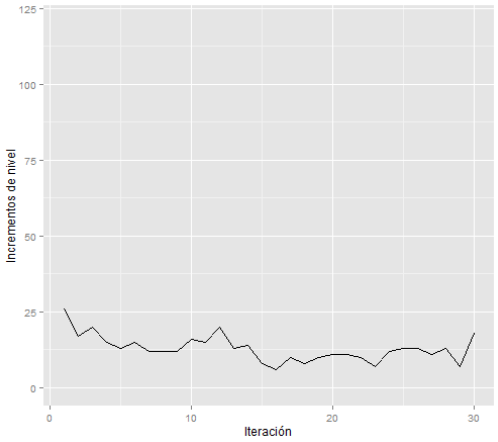


Ilustración 73 Gráfica Número de incrementos de nivel por ciclo – Azar

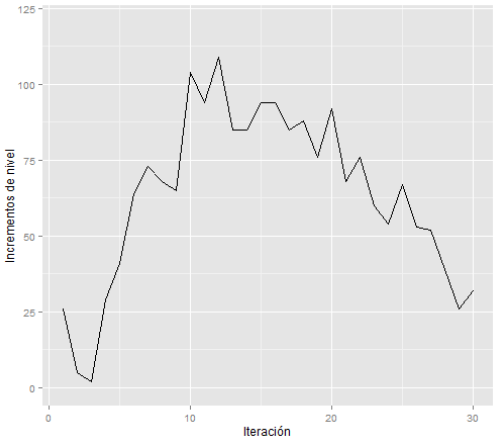


Ilustración 75 Gráfica Número de incrementos de nivel por ciclo – Inversiones

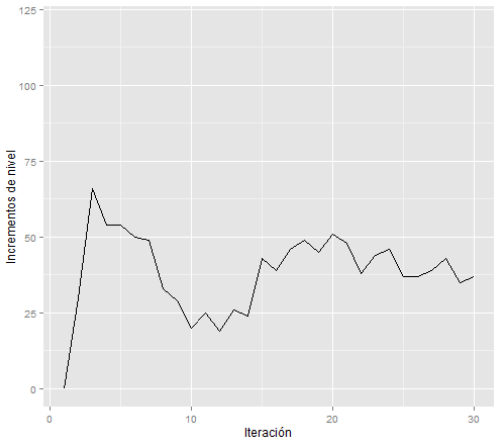


Ilustración 74 Gráfica Número de incrementos de nivel por ciclo – Experto

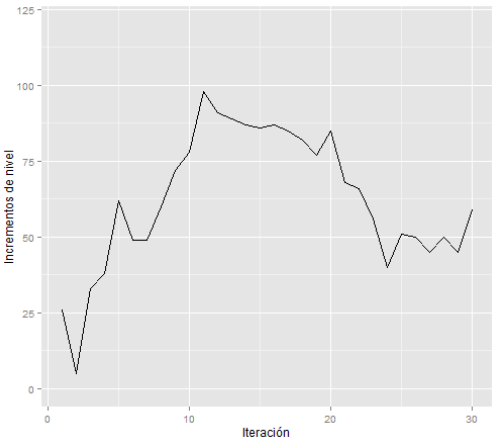


Ilustración 76 Gráfica Número de incrementos de nivel por ciclo – Usuarios

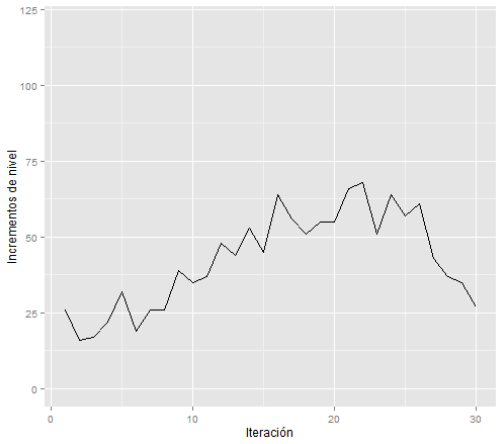


Ilustración 77 Gráfica Número de incrementos de nivel por ciclo – Problemas

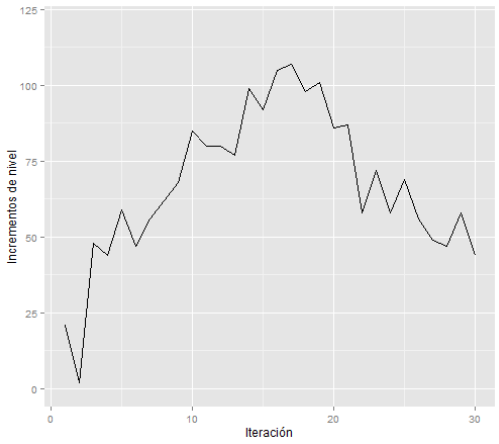


Ilustración 78 Gráfica Número de incrementos de nivel por ciclo – SVD

Número de usuarios que completaron los problemas

Para esta variable solo el recomendador Experto tuvo usuarios con todos los problemas, en la ilustración 79 se muestra la gráfica del número de usuarios que completaron todos los problemas.

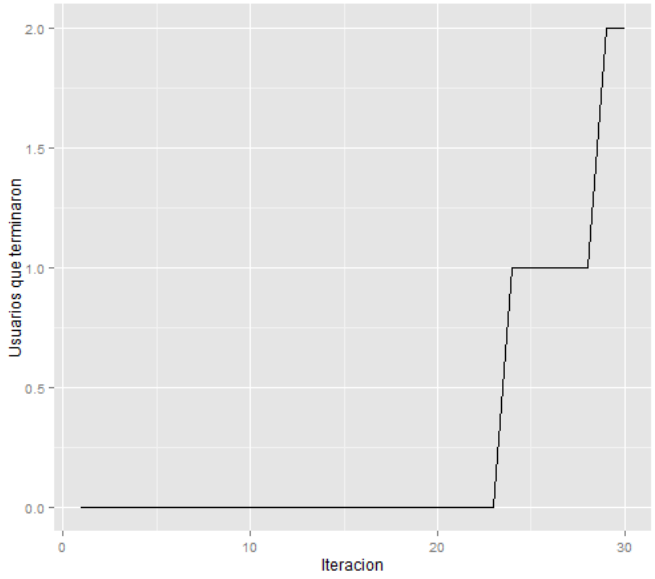


Ilustración 79 Número de usuarios que completaron los problemas - Experto

Número de usuarios rendidos

En las ilustraciones 80-85 se muestran el total de usuarios rendidos de cada uno de los recomendadores. Los mejores recomendadores son aquellos que cuentan con el menor número de usuarios rendidos.

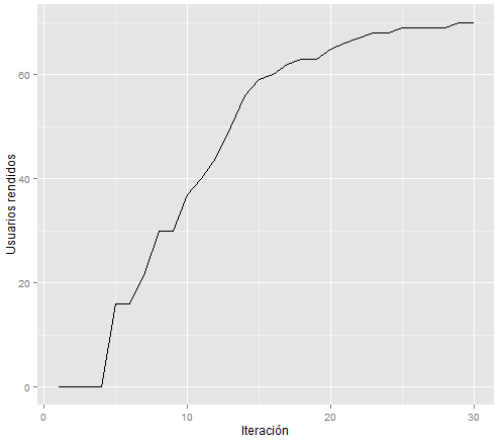


Ilustración 80 Gráfica Número de usuarios rendidos – Azar

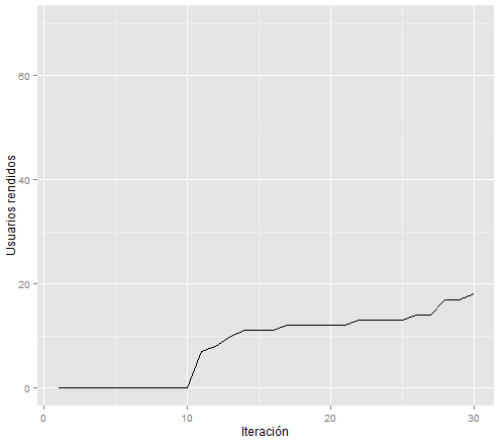


Ilustración 81 Gráfica Número de usuarios rendidos – Experto

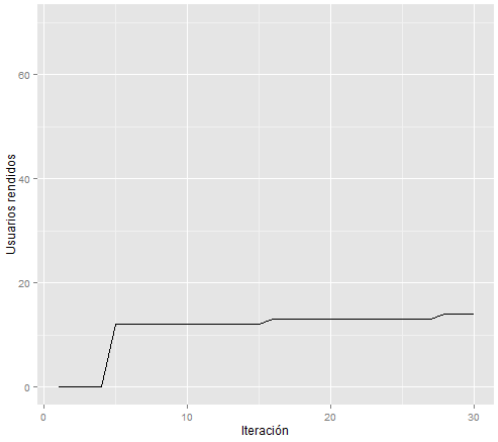


Ilustración 82 Gráfica Número de usuarios rendidos – Inversiones

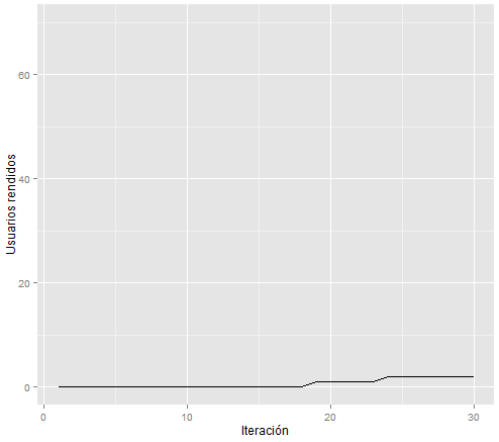


Ilustración 83 Gráfica Número de usuarios rendidos – Usuarios

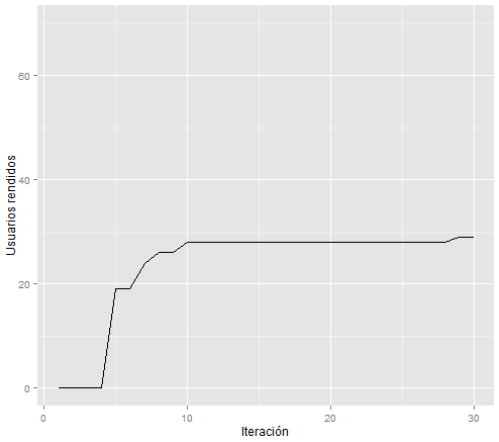


Ilustración 84 Gráfica Número de usuarios rendidos – Problemas

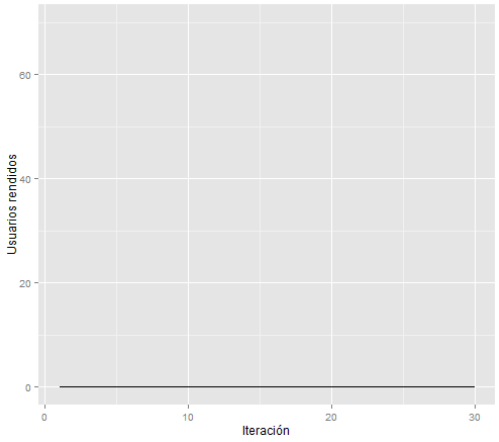


Ilustración 85 Gráfica Número de usuarios rendidos – SVD

Número de veces que se utilizó coldStart

Cuando el recomendador no tiene suficiente información para dar una recomendación puede recurrir a un segundo recomendador las gráficas 86-89 muestran el número de veces que fue llamado este segundo recomendador, para el caso del recomendador al azar y experto no es posible tener un segundo recomendador.

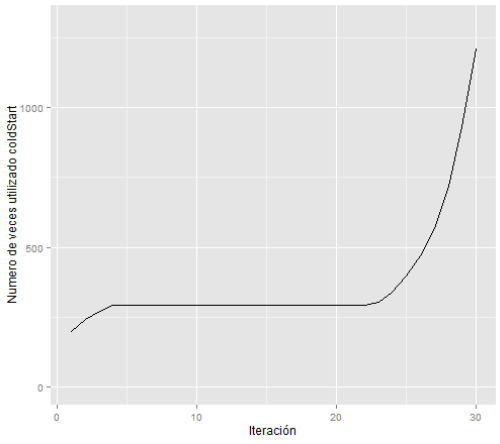


Ilustración 86 Gráfica Número de veces que se utilizó coldStart – Inversiones

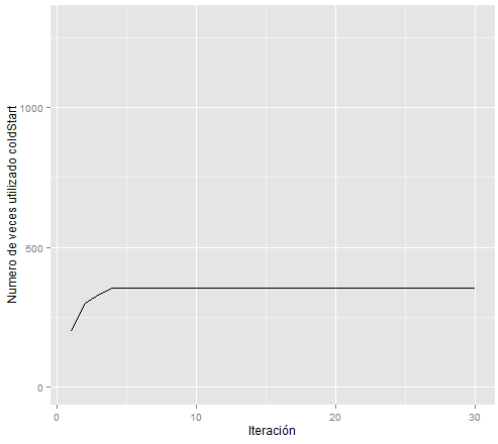


Ilustración 88 Gráfica Número de veces que se utilizó coldStart – Problemas

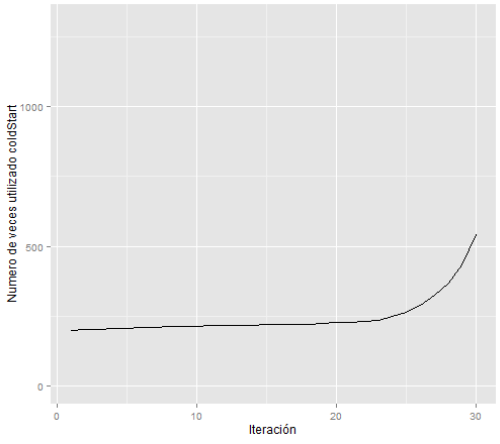


Ilustración 87 Gráfica Número de veces que se utilizó coldStart – Usuarios

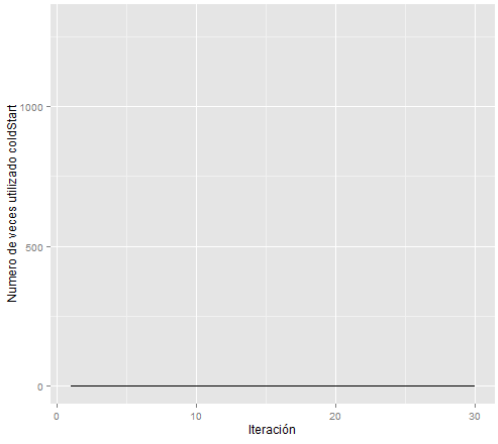


Ilustración 89 Gráfica Número de veces que se utilizó coldStart – SVD

Número de veces que se utilizó coldStar por ciclo

Para ver con detalle cuando se utilizó el recomendador las siguientes gráficas (Ilustración 90-93) muestran el número de veces que se utilizó en cada ciclo.

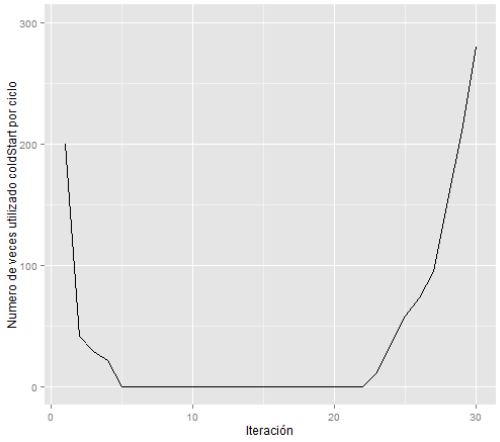


Ilustración 90 Gráfica Número de veces que se utilizó coldStar por ciclo – Inversiones

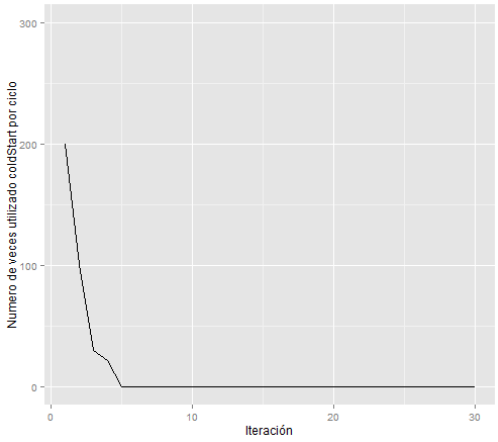


Ilustración 92 Gráfica Número de veces que se utilizó coldStar por ciclo – Problemas

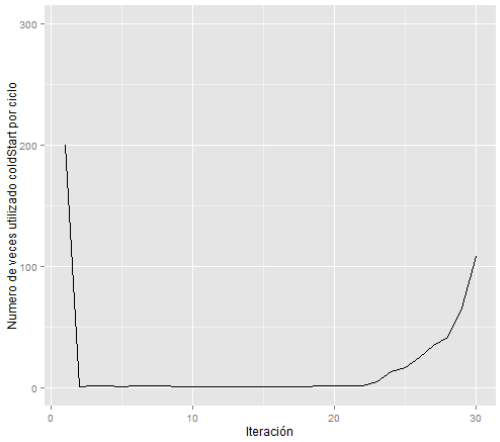


Ilustración 91 Gráfica Número de veces que se utilizó coldStar por ciclo – Usuarios

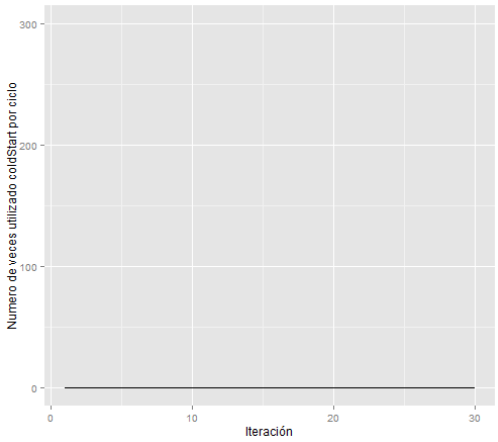


Ilustración 93 Gráfica Número de veces que se utilizó coldStar por ciclo – SVD

Precisión de recomendaciones

La precisión es el número de problemas resueltos entre el número de problemas recomendados. Las ilustraciones 94-99 muestran la precisión en cada ciclo de cada uno de los recomendadores.

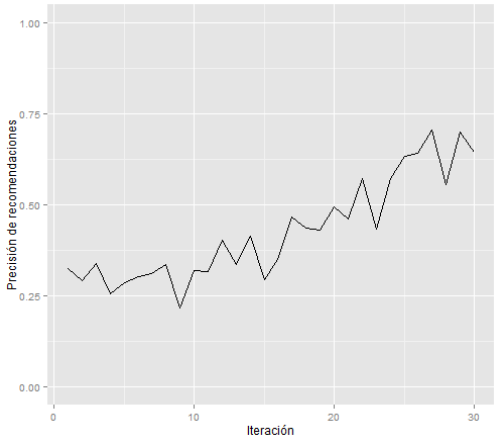


Ilustración 94 Gráfica Precisión de recomendaciones – Azar

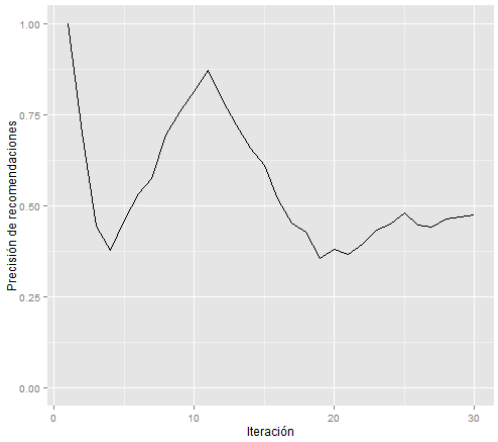


Ilustración 95 Gráfica Precisión de recomendaciones – Experto

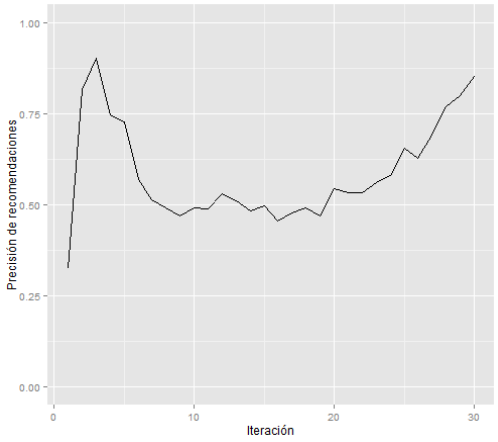


Ilustración 96 Gráfica Precisión de recomendaciones – Inversiones

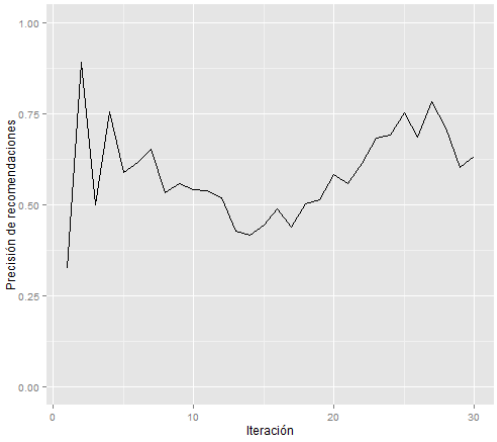


Ilustración 97 Gráfica Precisión de recomendaciones – Usuarios

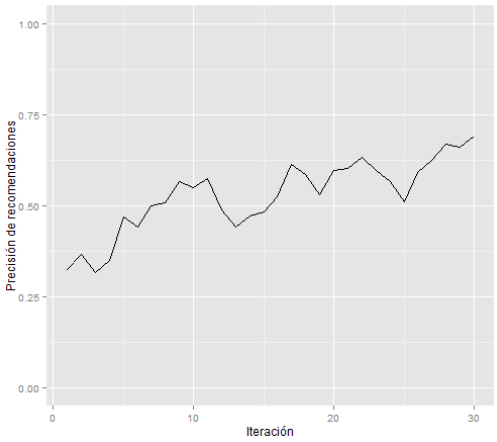


Ilustración 98 Gráfica Precisión de recomendaciones – Problemas

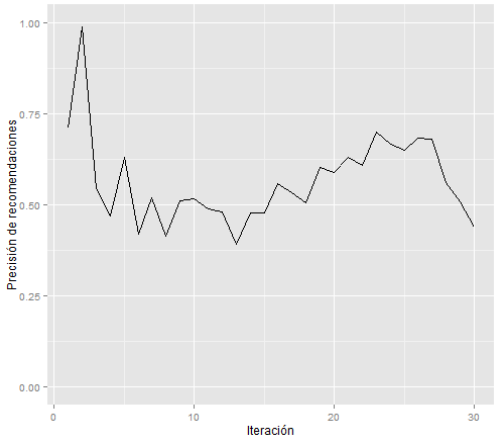


Ilustración 99 Gráfica Precisión de recomendaciones – SVD

Precisión de recomendaciones por el logaritmo del número de recomendaciones

Como también es importante el número de recomendaciones que da se multiplica la precisión por el logaritmo del número de recomendaciones para obtener una medida que toma en consideración el número de recomendaciones dadas. En las ilustraciones 100-105 se muestran las gráficas de esta medida de cada uno de los recomendadores en cada ciclo.

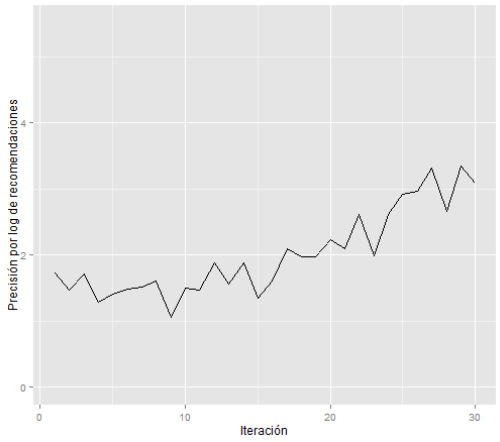


Ilustración 100 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – Azar

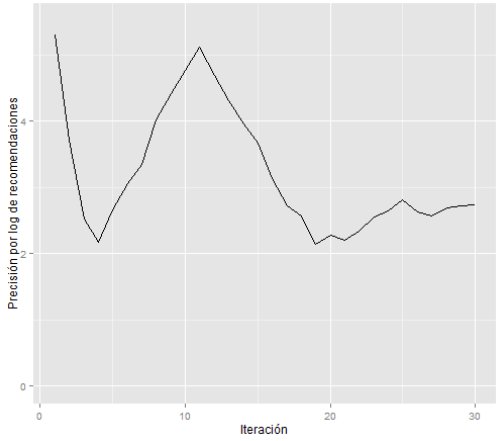


Ilustración 101 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – Experto

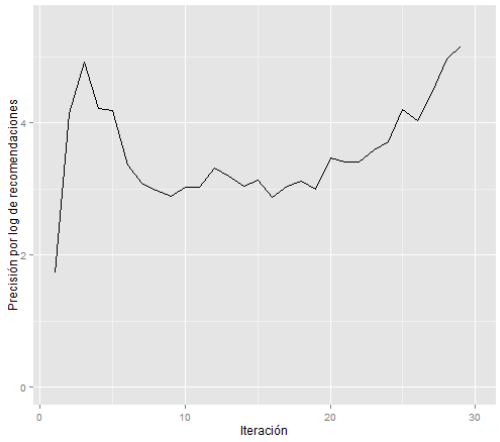


Ilustración 102 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – Inversiones

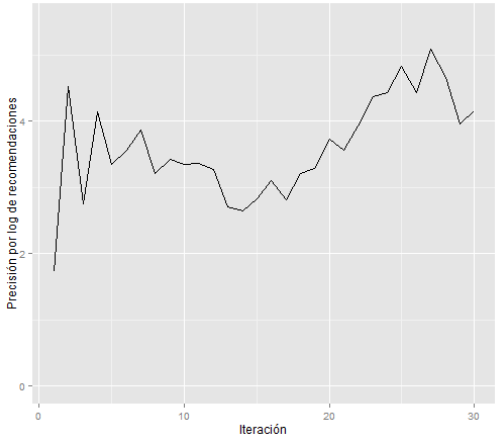


Ilustración 103 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – Usuarios

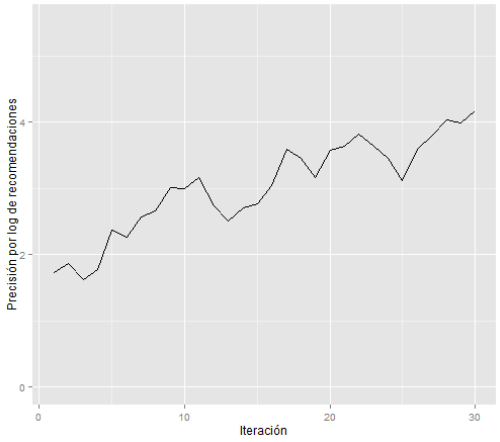


Ilustración 104 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – Problemas

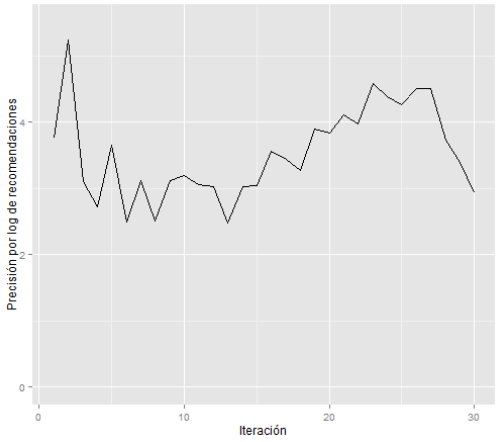


Ilustración 105 Gráfica Precisión de recomendaciones por el logaritmo del número de recomendaciones – SVD

Tiempo tomado en la función realiza análisis

Las siguientes gráficas (Ilustraciones 106-109) muestran el tiempo tomado en cada ciclo por la función realiza análisis en milisegundos. No aparecen los recomendadores Azar ni Experto porque estos sus tiempos son cercanos a cero comparados con el tiempo que toman el resto de los recomendadores.

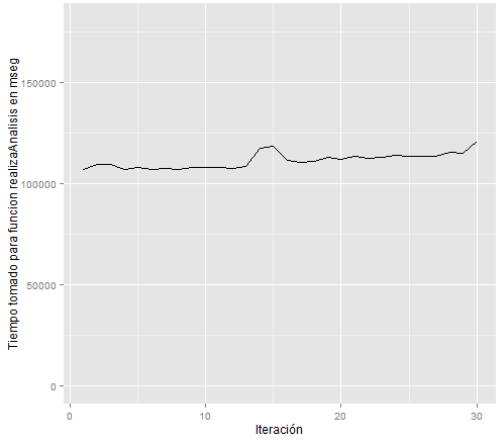


Ilustración 106 Gráfica Tiempo tomado en la función realiza análisis – Inversiones

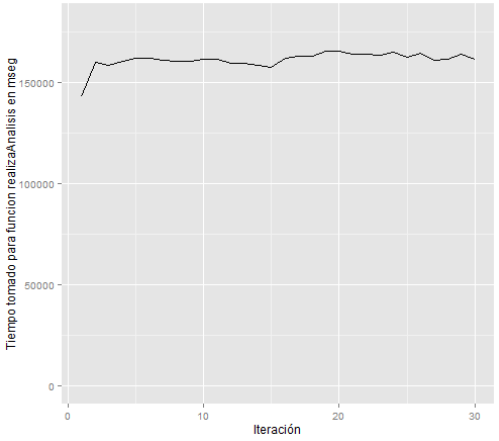


Ilustración 107 Gráfica Tiempo tomado en la función realiza análisis – Usuarios

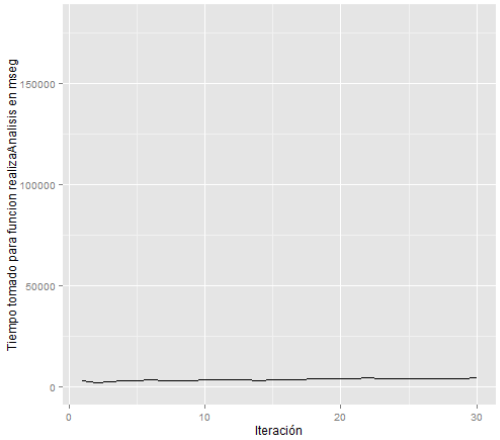


Ilustración 108 Gráfica Tiempo tomado en la función realiza análisis – Problemas

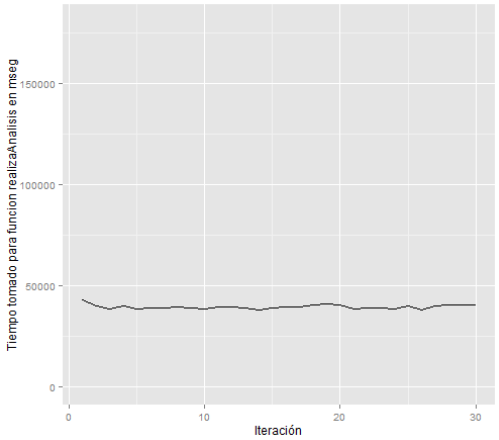


Ilustración 109 Gráfica Tiempo tomado en la función realiza análisis – SVD

Tiempo promedio tomado en dar una recomendación

Para las recomendaciones se obtuvo el tiempo promedio tomado en cada recomendación durante cada ciclo. Las siguientes gráficas (Ilustraciones 110-115) muestran el tiempo promedio tomado por cada recomendador durante cada ciclo en milisegundos.

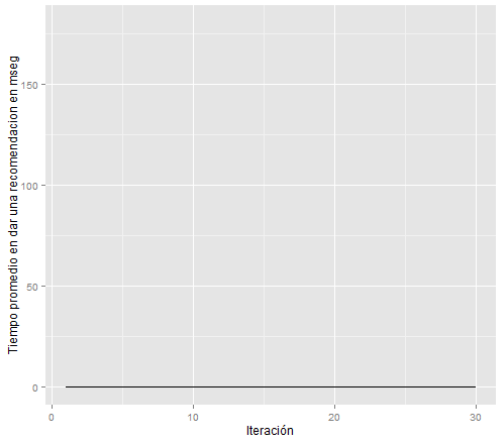


Ilustración 110 Gráfica Tiempo promedio tomado en dar una recomendación – Azar

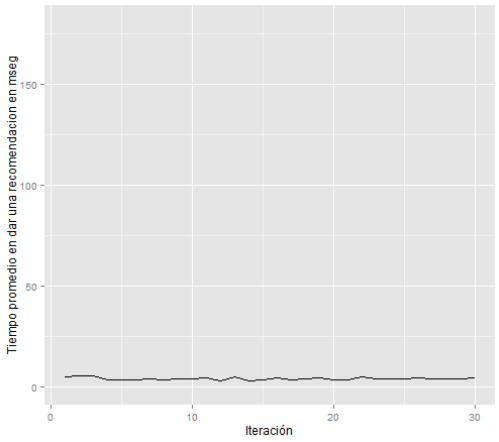


Ilustración 111 Gráfica Tiempo promedio tomado en dar una recomendación – Experto

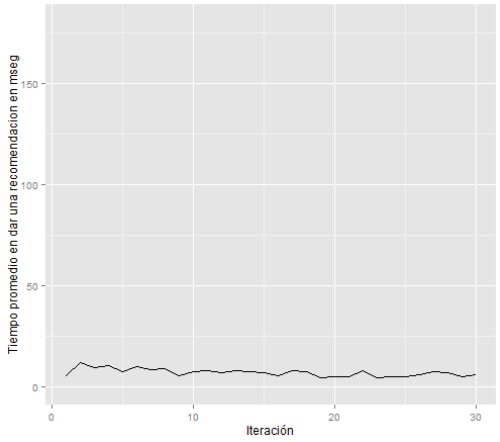


Ilustración 112 Gráfica Tiempo promedio tomado en dar una recomendación – Inversiones

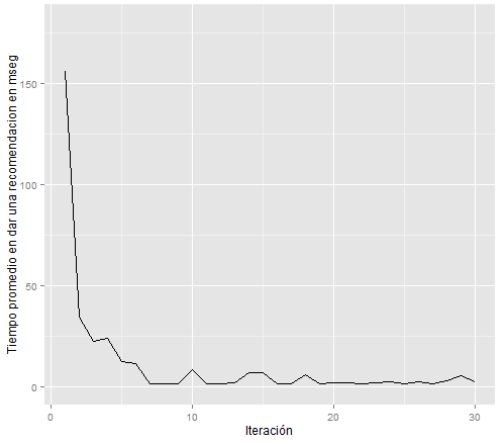


Ilustración 114 Gráfica Tiempo promedio tomado en dar una recomendación – Problemas

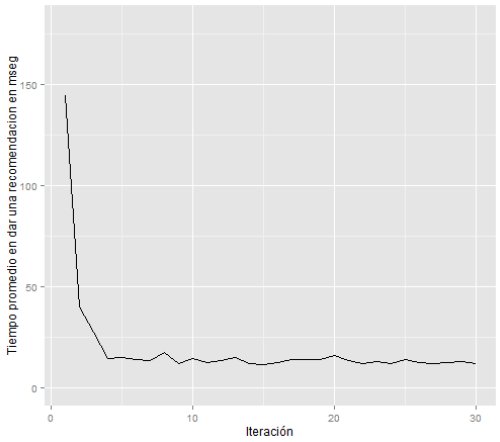


Ilustración 113 Gráfica Tiempo promedio tomado en dar una recomendación – Usuarios

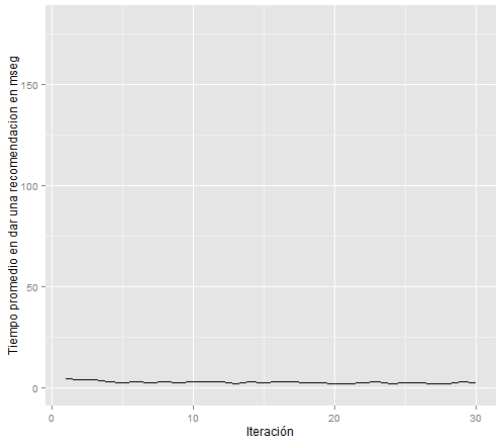


Ilustración 115 Gráfica Tiempo promedio tomado en dar una recomendación – SVD

Root Mean Square Error

Para poder ajustar la razón de aprendizaje se utilizó esta gráfica, si la razón era muy alta el RMSE no converge si es muy baja el RMSE converge muy lento. La ilustración 116 muestra la gráfica del RMSE en cada iteración de aprendizaje durante la primera vez que se realiza el análisis. En la gráfica se puede observar cuando se empieza a entrenar una nueva característica.

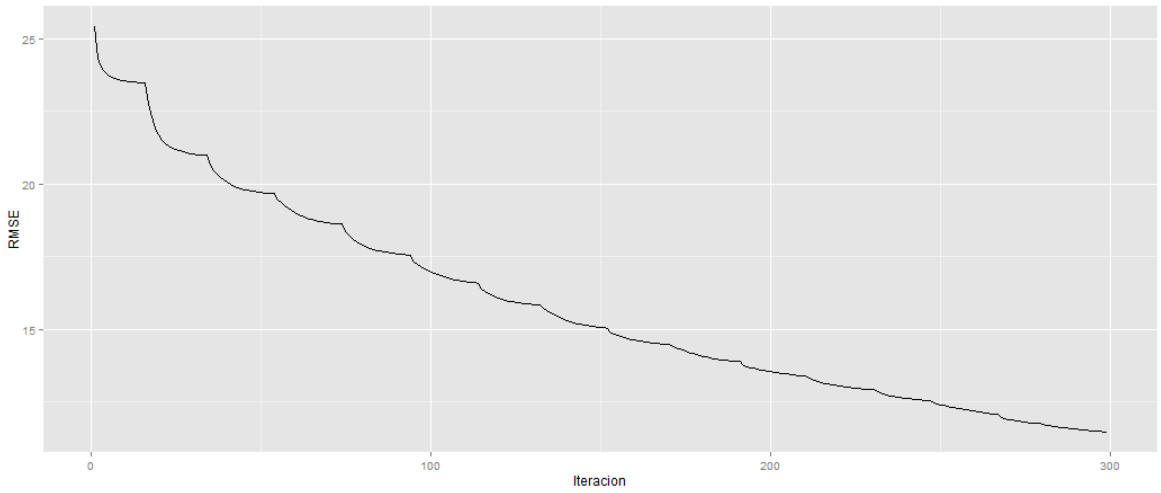


Ilustración 116 Gráfica Root Mean Square Error entrenamiento SVD

Análisis de sensibilidad parámetros de usuarios

Para el análisis de sensibilidad se utilizaron los mismos valores de los parámetros que en la ejecución de las pruebas a excepción del parámetro que se esté analizando. En las ilustraciones 117 y 118 se pueden observar los resultados obtenidos.

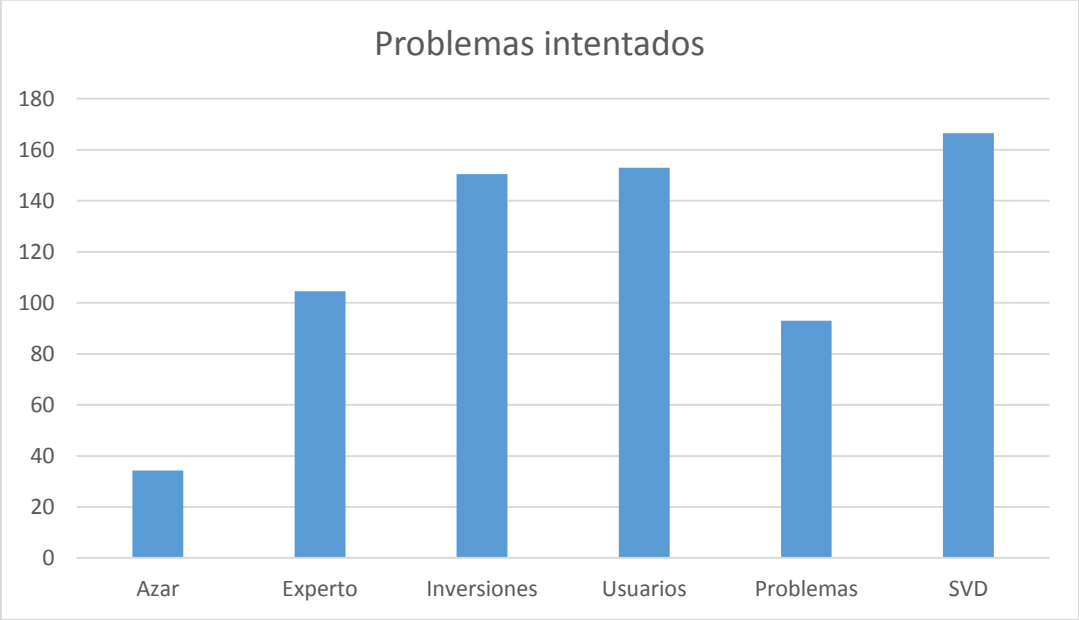


Ilustración 117 Gráfica el número de problemas intentados por usuario para cada uno de los algoritmos de recomendación.

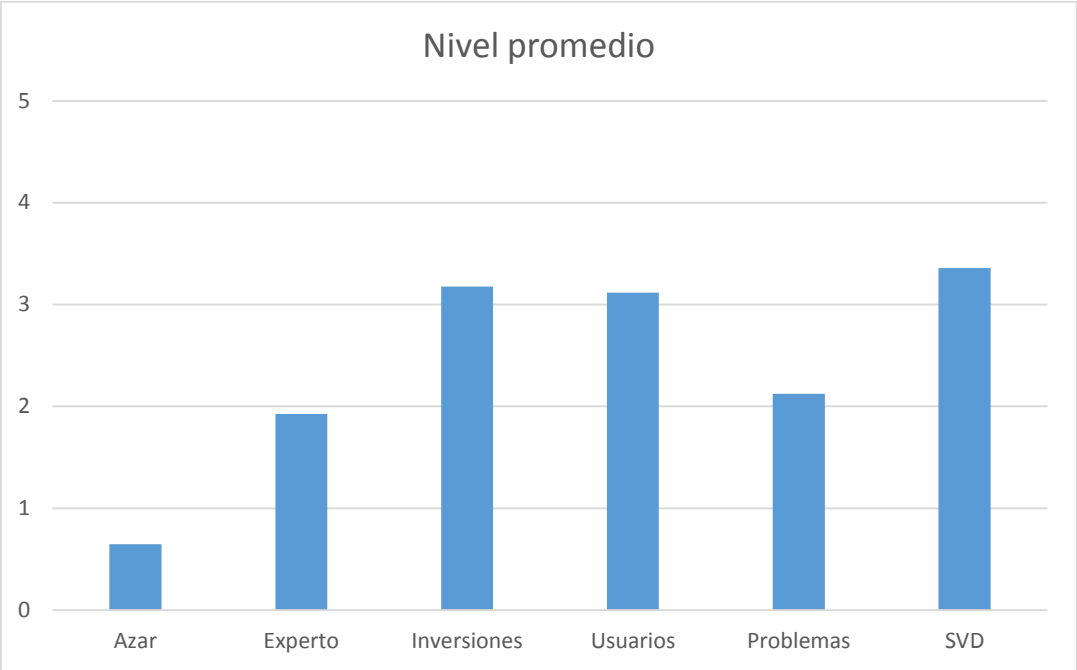


Ilustración 118 Gráfica nivel promedio de los alumnos para cada uno de los algoritmos de recomendación

aPositiva

Para hacer el análisis de sensibilidad del parámetro aPositiva, se varió su valor en el rango de 0 a 2. En la Ilustración 119 se puede observar el comportamiento de cada uno de los algoritmos cuando se varía este valor donde 0 significa que la motivación no se incrementa con los problemas resueltos al incrementar el valor indica que el usuario es motivado más por los éxitos al resolver algún problema. Entre mayor sea el número de problemas intentados quiere decir que los alumnos se mantuvieron más tiempo resolviendo problemas lo cual es el objetivo de esta tesis. En la gráfica podemos observar que los algoritmos que constantemente cumplen el objetivo son: factorización de matrices, similitud de usuarios e inversiones.

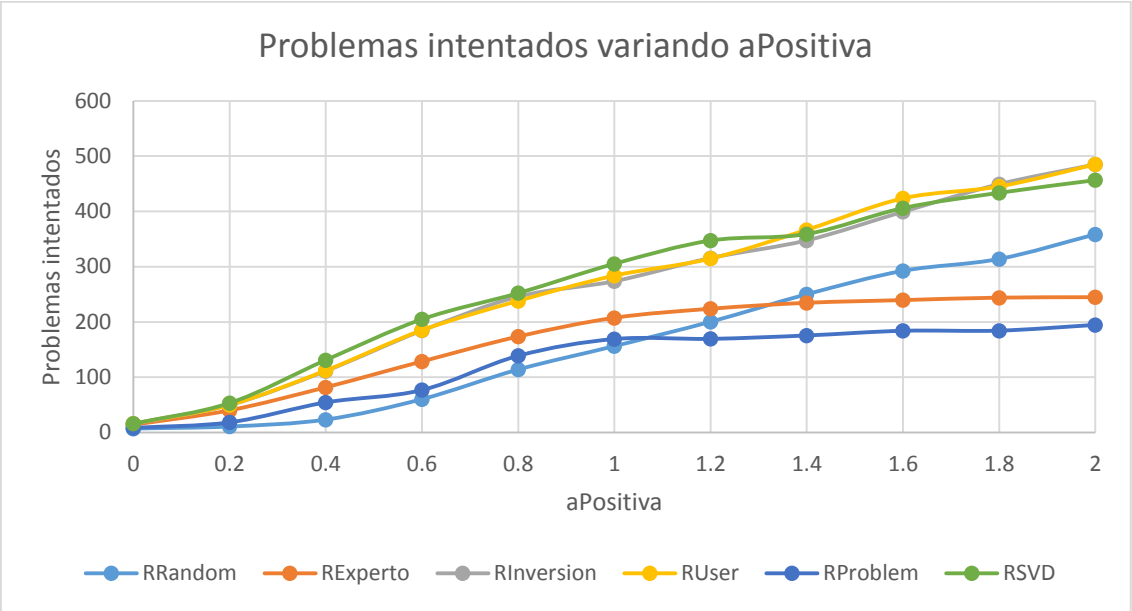


Ilustración 119 Gráfica Problemas intentados por usuario variando el valor aPositiva

En la ilustración 120 se observa el nivel promedio de los usuarios se observa que también los algoritmos de factorización de matrices, similitud de usuarios mostraron el mejor desempeño.

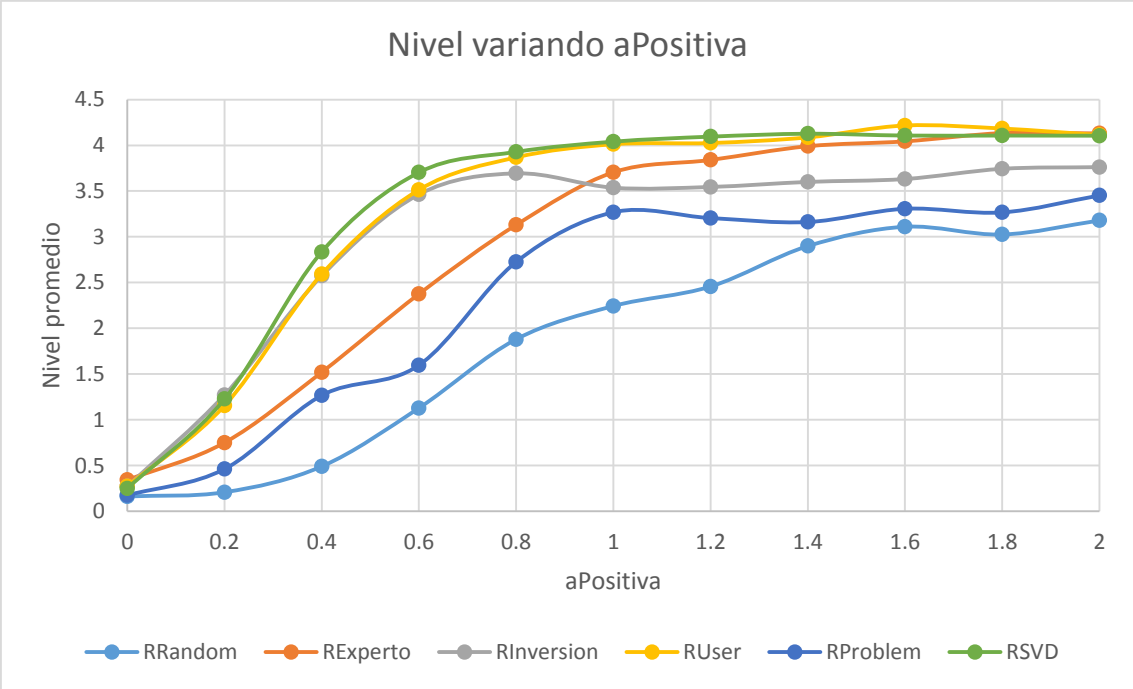


Ilustración 120 Gráfica Nivel promedio variando el valor de aPositiva

aNegativa

El parámetro aNegativa indica que tan sensible es el alumno a los fallos donde 0 indica que fallar no disminuye la motivación y entre mayor sea este valor el alumno es desmotivado en mayor medida por los errores cometidos. En la Ilustración 121 se muestran los resultados al variar este valor y se

puede observar que los algoritmos que se mantienen dando los mejores resultados son factorización de matrices, inversiones y similitud de usuario.

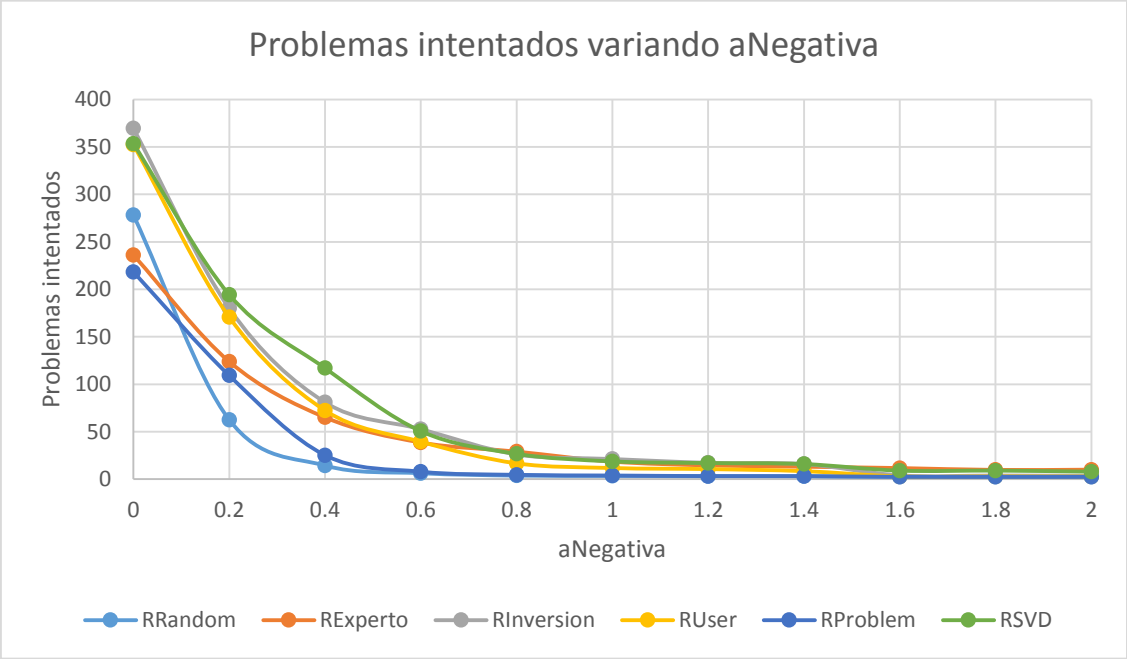


Ilustración 121 Gráfica Problemas intentados por usuario variando el valor aNegativa

Al igual que los problemas intentados el nivel promedio disminuye, al incrementar el valor de aNegativa.

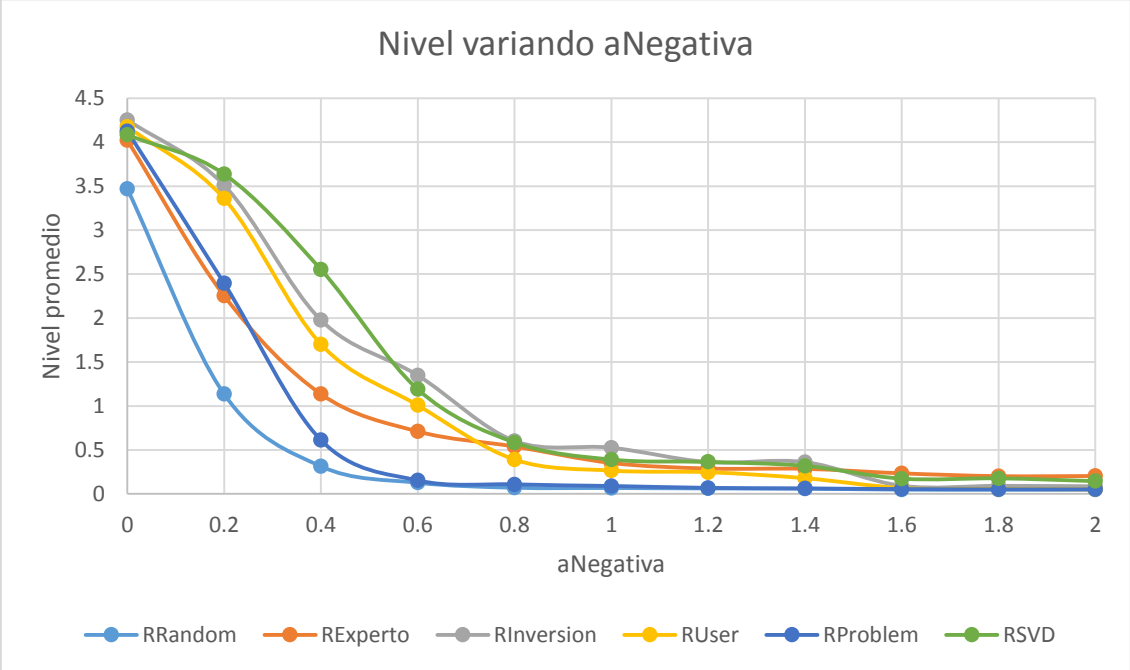


Ilustración 122 Gráfica Nivel promedio variando el valor de aNegativa

Motivación inicial

La motivación inicial ayuda a que se puedan tener varios errores en las primeras recomendaciones. En la ilustración 123 se puede observar que la mayoría de los recomendadores no son muy sensibles a la variación de esta variable a excepción del algoritmo basado en similitud de problemas el cual alcanza un rendimiento similar al experto después de cierto valor de motivación inicial.

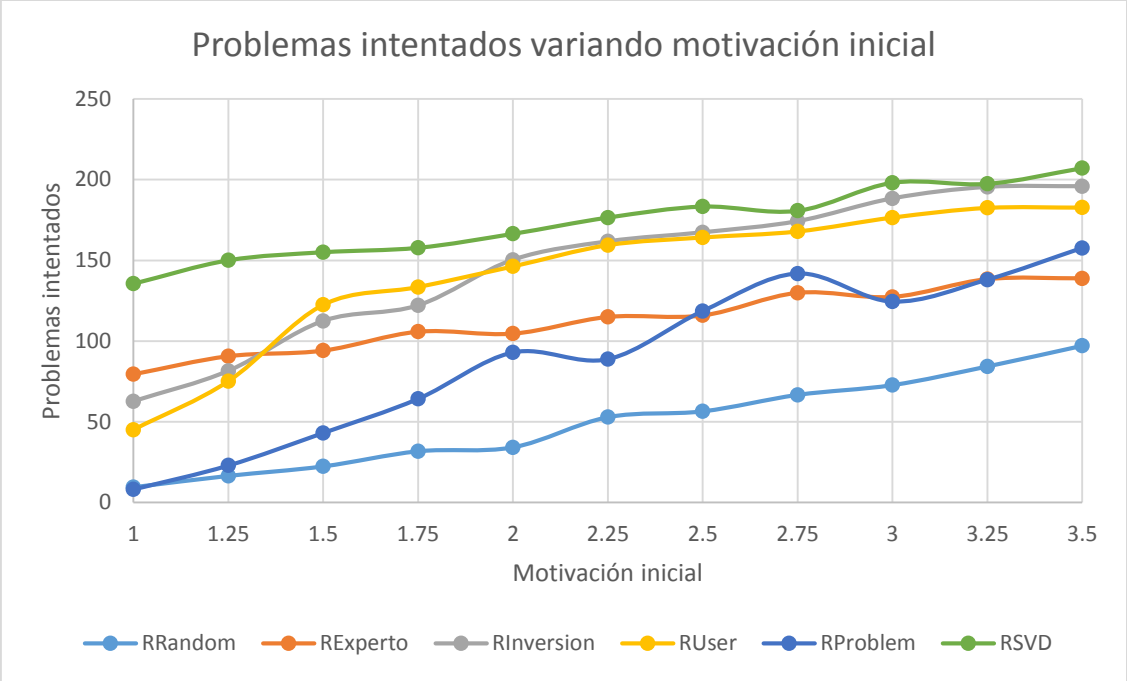


Ilustración 123 Problemas intentados por usuario variando el valor motivación inicial

En la ilustración 124 se muestra el comportamiento del nivel promedio, el cual tiene un comportamiento similar al observado por los problemas intentados. Se puede observar que las recomendaciones basadas en similitud de usuarios e inversiones necesitan una motivación inicial de al menos 1.5 para poder tener un rendimiento como la factorización de matrices.

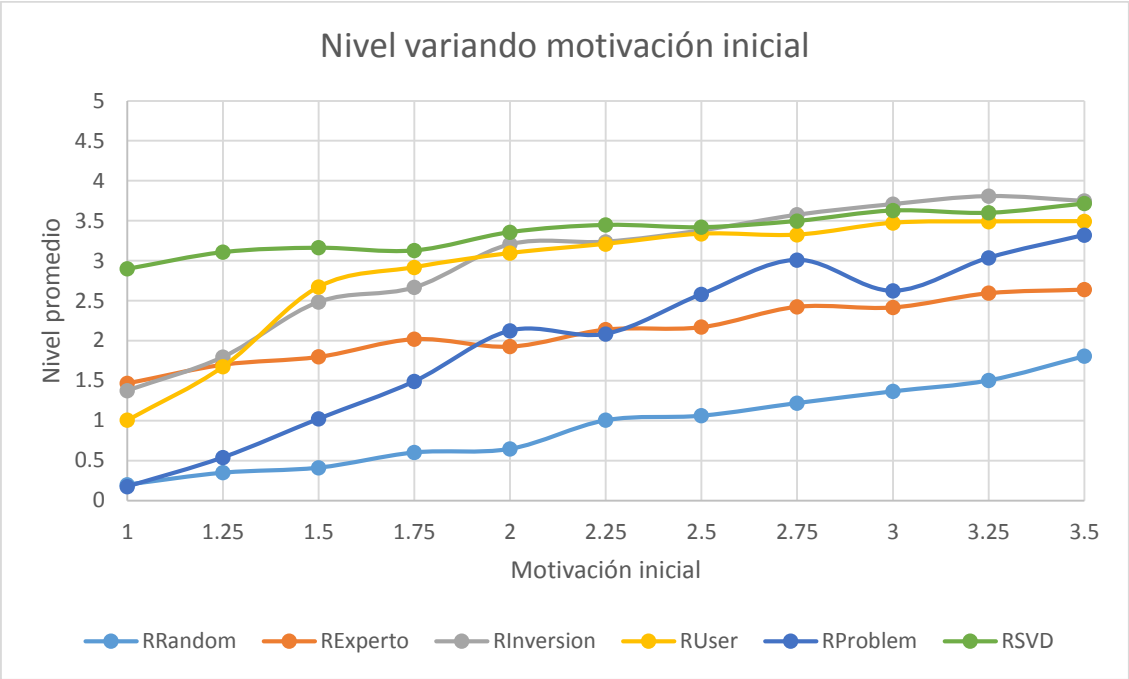


Ilustración 124 Gráfica Nivel promedio variando el valor de motivación inicial

fFacilidad

El factor de facilidad determina una penalización al incremento de motivación en caso de que la dificultad del problema resuelto difiere del nivel que tiene el usuario. Cuando fFacilidad tiene un valor igual a 0 no hay penalización y mientras mayor sea el valor de fFacilidad mayor es la penalización de acuerdo a la diferencia entre niveles.

En la ilustración 125 se muestra que al incrementar este factor todos los algoritmos reducen el número de problemas intentados, el algoritmo que parece más afectado por esto es el basado en un experto.

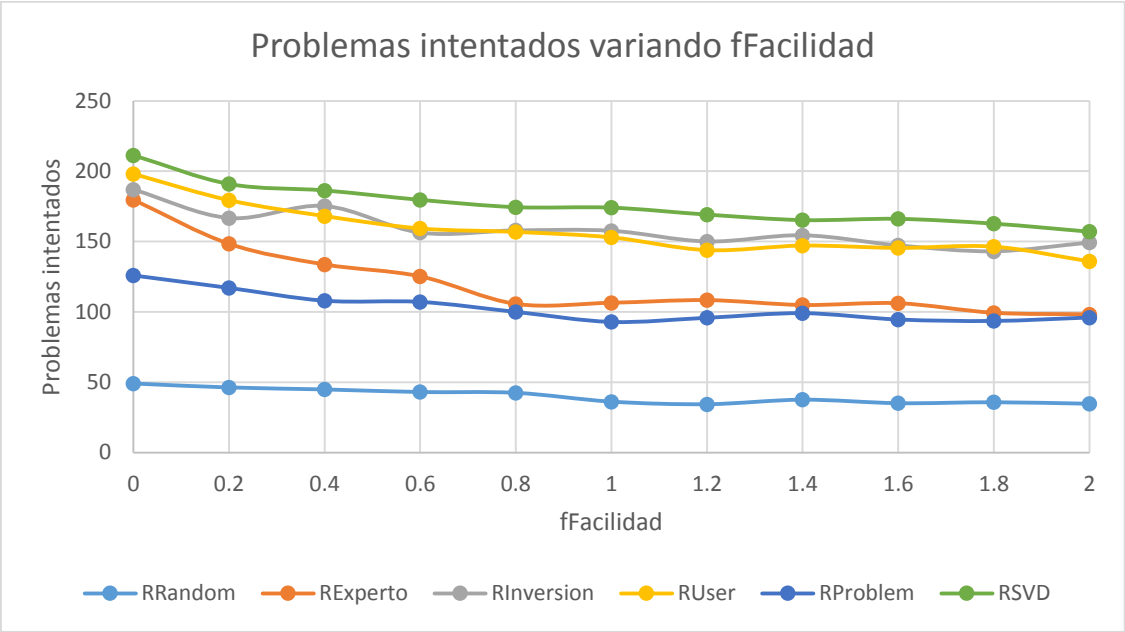


Ilustración 125 Gráfica Problemas intentados por usuario variando el valor fFacilidad

En la ilustración 126 se muestra el nivel promedio variando el factor de facilidad y se puede observar que el cambio en esta variable no afecta el rendimiento en la mayoría de los algoritmos excepto en las recomendaciones basadas en un experto.

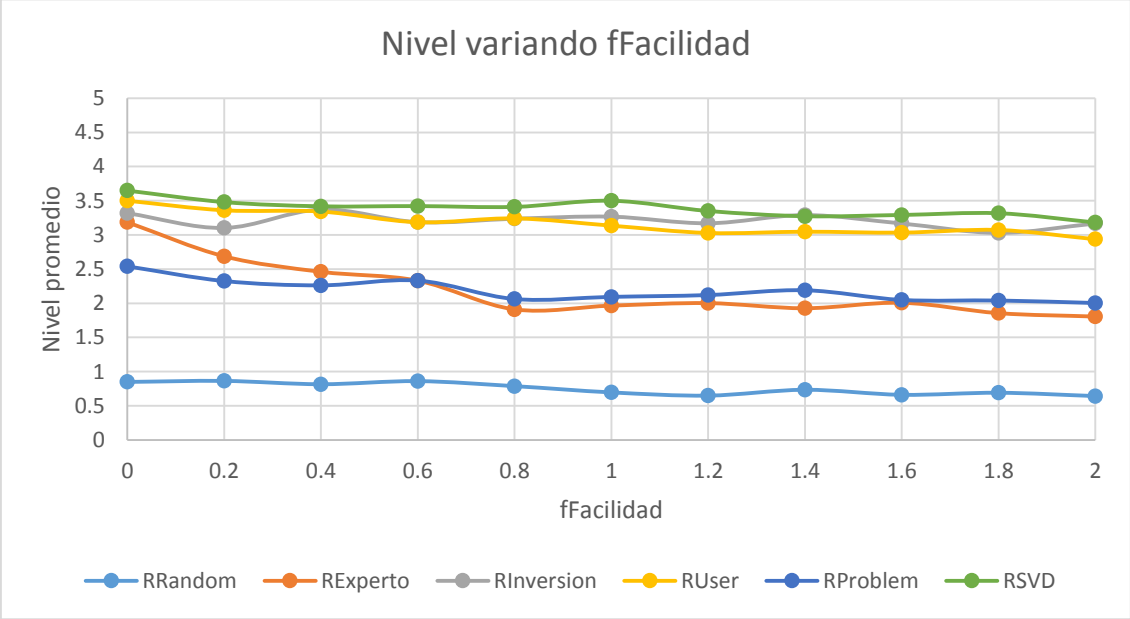


Ilustración 126 Gráfica Nivel promedio variando el valor de fFacilidad

Capítulo 6: Conclusiones, aportaciones y trabajo futuro

Conclusiones

En este documento se detalló el proceso para la evaluación de los 6 modelos de recomendación propuesto, para ello se especificó una simulación y las variables que se tomaron en cuenta para compararlos. La simulación utiliza los datos registrados por la página de Karelotitlán de tal manera que el proceso tenga características similares a un juez en línea. A partir del análisis realizado se puede concluir:

- El orden de los problemas puede afectar la motivación de los usuarios: bajo el modelo de usuarios propuesto, recomendar constantemente problemas que el usuario no puede resolver lleva a que el usuario desista de continuar y por lo tanto no invierte el tiempo necesario para mejorar sus habilidades de igual manera proponer problemas de una gran dificultad pero que se pueden resolver ayudan a que el usuario mejore sus habilidades.
- Se pueden utilizar filtros colaborativo para producir recomendaciones que son mejores que el azar y una lista otorgada por un experto: Por los resultados obtenidos se puede decir que los recomendadores basado en inversiones, basado en similitud de usuarios y basado en la factorización de usuarios presentan un mayor número de problemas intentados y mayor número de incrementos de nivel que los recomendadores al azar y basado en un experto.
- El recomendador basado en número de inversiones, similitud de usuarios y en la factorización de matrices, mostraron que pueden incrementar significativamente el nivel de los usuarios y mantenerlos motivados para que resuelvan más problemas: Estos tres recomendadores mostraron mejores resultados que el resto de los recomendadores en el número de problemas intentados y en el número de incrementos de nivel.
- En caso de no contar con un experto para sugerir una secuencia de problemas a resolver, es posible utilizar otros modelos de recomendación (SVD, Inversiones, Usuarios) que hacen un trabajo igual o superior: Estos recomendadores fueron probados utilizando el recomendador al azar cuando no poseían suficiente información, esto muestra que pueden ser utilizados sin la necesidad de una lista previa.

Aportación del trabajo

Este trabajo tuvo las siguientes aportaciones:

- Planteamiento de un nuevo problema de recomendación: En esta tesis se especificó el problema de dar recomendaciones de tal maneara que el usuario pueda incrementar sus habilidades y no desista de resolver problemas.
- Descripción de una simulación para evaluar modelos de recomendación: Se planteó una simulación del proceso de entrenamiento para facilitar las pruebas y también poder obtener algunas mediciones con las cuales comparar diferentes recomendadores.
- Rediseño de tres modelos de recomendación utilizados para música y películas: Se tomaron tres de los recomendadores más usados (usuarios, problemas y SVD) para predecir preferencias de música, películas, noticias, etc. se adaptaron para poder dar respuesta al nuevo problemas planteado y se implementaron para hacer pruebas de estos en la simulación.
- Propuesta de un nuevo modelo de recomendación: se propuso un nuevo recomendador (inversiones) diseñado específicamente para este problema.

- Comparación de 6 modelos de recomendación: se implementó la simulación descrita y 6 recomendadores, estos se simularon para poder determinar qué características tenía cada uno.

Trabajo Futuro

Aun se puede realizar más trabajos para aumentar el impacto de los jueces en línea, algunas de mis propuestas para seguir trabajando son:

- Implementación de la recomendación en OmegaUp: OmegaUp es otro juez en línea creado por mexicanos, este juez en línea cuenta con el detalle de todos los envíos pero no tiene identificado que tema abarca cada problema. Cualquier persona puede subir problemas a esta plataforma lo cual hace muy difícil tener control de la calidad de los problemas. Es por eso que para esta página se está convirtiendo indispensable poder contar con un recomendador.
- Etiquetado automático de problemas por tema: para este trabajo se tenía previamente etiquetados los problemas por tema esta información es de gran utilidad para decidir qué problemas resolver para el caso de jueces en línea como OmegaUp se busca una forma de etiquetar los problemas. Pareciera ser que se podría etiquetar por medio de votaciones o por medio de análisis de los resultados de problemas.
- Etiquetado automático de problemas por dificultad: Para poder generar los datos de simulación se le asignó una dificultad a cada problema de forma manual, un proceso interesante a investigar seria como etiquetar cada problema de forma automática. Un primer acercamiento es ordenar de acuerdo al número de envíos aceptados entre el número de envíos, pero esta estadística resulta no ser suficiente para emitir una dificultad específica para cada usuario, lo cual vuelve interesante en pensar en mejor esto.

Tabla de ilustraciones

Bibliografía

[1] P. Alvarez y L. R. Squire, «Memory consolidation and the medial temporal lobe: A simple network model,» *Proceedings of the National Academy of Sciences of the United States of America - Neurobiology,* vol. 91, pp. 7041-7045, July 1994.

[2] C. M. Alberini, «Mechanisms of memory stabilization: are consolidation and reconsolidation similar or distinct processes?,» *Trends in Neurosciences,* vol. 28, nº 1, Enero 2005.

[3] M. C. Inda, E. V. Muravieva y C. M. Alberini, «Memory Retrieval and the Passage of Time: From Reconsolidation and Strengthening to Extinction,» *The Journal of Neuroscience,* pp. 1635-1643, 2 Febrero 2011.

[4] International Olympiad in Informatics, «IOI - Past, Present, Future,» International Olympiad in Informatics, Enero 2008. [En línea]. Available: http://www.ioinformatics.org/history.shtml. [Último acceso: Mayo 2014].

[5] International Olympiad in Informatics, «International Olympiad in Informatics - Statistics,» [En línea]. Available: http://stats.ioinformatics.org/olympiads/. [Último acceso: 03 02 2015].

[6] Comité Mexicano de Informática, «Olimpiada Mexicana de Informática,» Enero 2014. [En línea]. Available: http://www.olimpiadadeinformatica.org.mx/. [Último acceso: Mayo 2014].

[7] J. R. M. S. Richard E. Pattis, Karel the robot (2nd ed.): a gentle introduction to the art of programming, Nueva York, Estados Unidos: John Wiley & Sons, Inc., 1995.

[8] General Assembly International Olympiad in Informatics, «International Olympiad in Informatics Regulations,» Agosto 2010. [En línea]. Available: http://www.ioinformatics.org/rules/reg10.pdf. [Último acceso: Mayo 2014].

[9] Comité Mexicano de Informática A. C., «Desempeño de México en las Olimpiadas Internacionales de Informática de 1998 a 2013,» Septiembre 2013. [En línea]. Available: http://www.olimpiadadeinformatica.org.mx/OMI/OMI/Documentos.aspx. [Último acceso: Mayo 2014].

[10] Comité Mexicano de Informática A. C., «Desempeño de México en las Olimpiadas Internacionales de Informática de 1998 a 2014,» Agosto 2014. [En línea]. Available: http://www.olimpiadadeinformatica.org.mx/OMI/OMI/Documentos.aspx. [Último acceso: 3 Febrero 2015].

[11] Sphere Research Labs, «Sphere Online Judge,» Sphere Research Labs, 2012. [En línea]. Available: http://www.spoj.com/. [Último acceso: 2014].

[12] Directi Group, «CodeChef,» Directi, 2009. [En línea]. Available: http://www.codechef.com/. [Último acceso: 2014].

[13] M. Mirzayanov, «Codeforces,» Mike Mirzayanov, 2014. [En línea]. Available: http://codeforces.com/. [Último acceso: 2014].

[14] topcoder, «TopCoder,» topcoder, 2014. [En línea]. Available: http://www.topcoder.com/. [Último acceso: Mayo 2014].

[15] USA Computing Olympiad, «USACO training,» USACO, [En línea]. Available: http://ace.delos.com/usacogate. [Último acceso: 2014].

[16] F. R. H. Cuadrilla, «Karelotitlán,» Félix Rafael Horta Cuadrilla, 2013. [En línea]. Available: http://www.cmirg.com/karelotitlan. [Último acceso: 2014].

[17] OmegaUp, «OmegaUp,» OmegaUp, 2014. [En línea]. Available: omegaup.com. [Último acceso: 2014].

[18] A. Adomavicius y A. Tuxhilin, «Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and possible extensions,» *IEEE Transactions on knowledge and data engineering,* vol. 17, nº 6, 2005.

[19] D. Jannach, M. Zanker, A. Felfernig y G. Friedrich, Recommender Systems, Julio: Cambridge University Press, 2013.

[20] D. Goldberg, D. Nichols, B. M. Oki y D. Terry, «Using Collaborative Filtering to Weave an Information Tapestry,» *Commun. ACM,* vol. 35, nº 12, pp. 61-70, Diciembre 1992.

[21] P. Resnick, N. Lacovou, M. Suchak, P. Bergstrom y J. Riedl, «GroupLens: An Open Architecture for Collaborative Filtering of Netnews,» de *CSCW '94*, Chapel Hill, North Carolina, USA, 1994.

[22] U. Shardanand y P. Maes, «Social Information Filtering: Algorithms for Automating "Word of Mouth",» de *CHI '95*, Denver, Colorado, USA, 1995.

[23] R. D. Burke, K. J. Hammond y B. C. Young, «Knowledge-Based Navigation of Complex nformation Spaces,» de *AAAI-96*, Portland, Oregon, 1996.

[24] B. M. Marlin y R. S. Zemel, «Collaborative Prediction and Ranking with Non-random Missing Data,» de *RecSys '09*, New York, New York, USA, 2009.

[25] J. L. Herlocker, J. A. Konstan, L. G. Terveen y J. T. Riedl, «Evaluating Collaborative Filtering Recommender Systems,» *ACM Trans. Inf. Syst.,* vol. 22, nº 1, pp. 5-53, Enero 2004.

[26] J. L. Herlocker, J. A. Konstan y J. Riedl, «Explaining collaborative filtering recommendations,» de *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, Philadelphia, Pennsylvania, USA, 2000.

[27] F. Robson, «Numbers Game,» The Sydney Morning Herald, 10 Agosto 2013. [En línea]. Available: http://www.smh.com.au/digital-life/digital-life-news/numbers-game-20130809-2r9yi.html. [Último acceso: 22 Abril 2015].

[28] T. Verhoeff, G. Horváth, K. Diks, G. Cormack y M. Forisek, «IOI Syllabus,» 20 Febrero 2013. [En línea]. Available: http://people.ksp.sk/~misof/ioi-syllabus/. [Último acceso: 22 Abril 2015].

[29] S. S. Stevens, «On the psychophysical law,» *Psychological Review,* vol. 64, nº 3, pp. 153-181, 1957.

[30] J. Kleinberg y E. Tardos, Algorithm Design, Boston: Pearson Education, 2006.

[31] X. Amatriain y J. Basilisco, «The Netflix Tech Blog,» Netflix, 6 Abril 2012. [En línea]. Available: http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html. [Último acceso: 24 Junio 2015].

[32] S. Funk, «The Evolution of Cybernetics A Journal,» Simon Funk, 11 Diciembre 2006. [En línea]. Available: http://sifter.org/~simon/journal/20061211.html. [Último acceso: 24 Junio 2015].