Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Reducción de especificaciones ecuacionales eliminando símbolos superfluos

Tesis presentada para optar al título de
Licenciatura en Ciencias de la Computación

Javier Martínez Viademonte

Director: Lic. Carlos López Pombo
Codirector: Prof. Dr. Marcelo Frias

Buenos Aires, febrero 2007

Resumen

Dado el alto costo computacional de las técnicas tradicionales de model-checking, para el caso en que el esfuerzo depende de la cantidad de símbolos involucrados se estudia el problema de reducción de especificaciones. Se modela el problema utilizando hipergrafos, se demuestra que la tarea es NP-completa, se implementan y evalúan estrategias exactas, golosas y heurísticas sobre distintas familias representativas de instancias, se desarrollan y analizan otras mejoras complementarias y se integra la herramienta resultante con *ReMo*, un model-checker relacional para fork algebra. Se concluye que la nueva herramienta alcanza su propósito.

1. Introducción

Cualquiera sea el modo en que se quiera utilizar una computadora para resolver problemas, será necesario describirle la tarea que se desea desempeñe. Dentro de las posibles descripciones, por un lado, se tiene que las más abstractas son aquellas que evitan analizar cada detalle y se concentran en los aspectos generales más importantes; por el otro, con énfasis en la precisión, las descripciones más formales permiten razonar sobre las propiedades del problema y su eventual solución, siendo susceptibles de ser procesadas por un programa.

Dado que las descripciones formales consiguen los más altos grados de confiabilidad, y aceptando la conveniencia de disponer de una visión abstracta del problema, es de interés estudiar formas abstractas y formales, como lo son las especificaciones algebraicas.

Además, dado que generalmente se acepta que, en el marco del proceso de desarrollo de software, los defectos se vuelven más graves y su corrección más costosa mientras más avanzada [GJM02] esté la construcción al momento de la detección, surge el interés por comprobar que se deriven una serie de propiedades deseadas a partir de una descripción temprana del problema.

Al combinar los formalismos abstractos y el interés por detectar rápidamente inconvenientes en las descripciones de los problemas, se abre el camino al uso de técnicas de model-checking [CGP00] (verificación de modelos) que, mediante herramientas, realizan búsquedas automáticas de inconsistencias.

Usualmente una descripción formal o especificación E consiste en un conjunto de propiedades o axiomas, que impone restricciones sobre el problema que está siendo modelado. Analizar el comportamiento de una solución a dicho problema consiste en verificar si un conjunto de propiedades P, que predica sobre este comportamiento, se logra satisfacer. Por consiguiente, verificar la especificación de un problema usando model-checking consiste en comprobar

$$E \models P,$$

donde $\models$ denota la relación de *satisfacción*. Al plantear una serie de propiedades se pretende verificar que, efectivamente, con la especificación se está modelando el problema a resolver, de manera que toda propiedad deseada sea consecuencia de las reglas básicas planteadas.

Sin embargo, aún disponiendo de una especificación adecuada, un importante obstáculo al model-checking es tratar con la explosión en el espacio de búsqueda, que ocurre cuando la cantidad de posibles estados para describir un problema crece bruscamente, en general asociada a sistemas con múltiples componentes

que interactúan entre sí, o a sistemas con estructuras de datos complejas que pueden asumir una gran cantidad de valores. Ya que las especificaciones de software suelen presentar este problema, comúnmente no pueden ser verificadas usando métodos model checking convencionales.

En particular, en esta tesis se desarrolla una herramienta, «*minime*», que se complementa con un model-checker ya existente, «*ReMo*» [GS05], para ayudarlo a alivianar la carga de la explosión en el espacio de búsqueda. *ReMo* es una herramienta desarrollada por el Grupo de Investigación en Métodos Formales Relacionales en el Departamento de Computación (DC) de la Facultad de Ciencias Exactas y Naturales (FCEN) de la Universidad de Buenos Aires (UBA).

2. Desarrollo

Esta sección concentra los aspectos teóricos del trabajo. Comienza describiendo detalladamente el problema concreto a abordar y, a continuación, plantea una interpretación abstracta apoyada sobre la teoría de grafos; se estudia la complejidad computacional del problema y se presentan algoritmos y propuestas para implementar una solución.

2.1. Problema concreto

Como contexto del problema hay que tener en cuenta al model-checker *ReMo*; una herramienta de validación que busca contraejemplos en los que se satisfaga una especificación pero se viole alguna propiedad esperada. Actualmente esa herramienta tiene un alcance limitado a raíz de la imposibilidad práctica de analizar especificaciones complejas, noción que demanda un criterio de medición de especificaciones para el cual conviene introducir varias explicaciones.

Empezando por lo básico, el model-checker es un programa que analiza especificaciones relacionales, cuyo lenguaje es una extensión de las *fork algebras* [Fri02]. A su vez, cada especificación está constituida por un conjunto de definiciones y fórmulas, en cualquiera de las cuales figuran símbolos de constantes.

El programa se encarga de construir asignaciones de valores a los símbolos de constantes sobre los que la especificación es interpretada. Si bien la cantidad de posibles asignaciones es potencialmente infinita, el model-checker fuerza una cota en el tamaño de los dominios; sin embargo, a pesar de la cota en las combinaciones, los reemplazos por lo general son muchos y analizarlos a todos consigue ocupar a casi cualquier computadora por más tiempo del deseable. Se dirá que la *complejidad* asociada a una especificación está dada por el esfuerzo que insumirá analizarla íntegramente y se la calculará mediante el producto de dos factores: el tiempo que insume analizar *una combinación* de valores y la *cantidad de combinaciones*.

No obstante, este trabajo no se interesa por estudiar el tiempo que insume analizar *una* combinación, por lo que supondrá que este factor puede ser omitido. El objetivo que se persigue consiste en disminuir la *cantidad* de combinaciones a evaluar; esto reduce el esfuerzo necesario para procesar una especificación, permitiendo el análisis de modelos más complejos.

Volviendo al criterio para medir especificaciones, cada símbolo de constante realiza un aporte a la complejidad; este aporte individual será llamado «peso» del símbolo y estará representado por un número mayor a 1 (uno). La complejidad total será sinónimo de la cantidad de combinaciones que evaluará el model-checker, calculada como el producto de todos los pesos.

Definición 1 (complejidad)**:** sean E una especificación, $consts$ una función que a partir de una especificación devuelve el conjunto de símbolos de constante involucrados y $peso_E$ una función que asigna un número real mayor a 1 (uno) a cada símbolo en $consts(E)$,

$$complejidad(E) = \prod_{c \in consts(E)} peso_E(c).$$

A su vez, el peso de cada símbolo de constante se calcula a partir de otros parámetros que también figuran en la especificación y, como conviene no dete-

nernos ahora en este punto, se deja el detalle del cálculo para el Apéndice B (Etiquetas). El subíndice de la función *peso* puede omitirse si la especificación de referencia se sobreentiende.

La idea principal para disminuir la cantidad de evaluaciones parte de tomar las fórmulas que figuran dentro de la especificación y buscar aquellas que permitan eliminar símbolos de constantes sin introducir otros cambios. Si encontramos estas fórmulas, la complejidad total disminuirá gracias a la quita de factores; además, como las fork algebras son una extensión ecuacional del cálculo relacional, las fórmulas que se buscan se saben ecuaciones.

Definición 2 (fórmula simple - f.s.)**:** es una ecuación $c = t(x_1, \ldots, x_n)$, con $t(\ldots)$ un término cualquiera, $\{x_1, \ldots, x_n\}$ el conjunto de símbolos de constantes que figuran en ese término y c un símbolo de constante que no figura en el conjunto. Se la abrevia «f.s.» y también es común darle nombre a la fórmula completa, digamos f, y notarlo $f : c = t$.

Las f.s. sugieren que cada aparición de la constante aislada puede ser reemplazada por el término, a lo largo de todo el conjunto de fórmulas que compone una especificación. Como, por la Definición 2, la constante no figura en el término, el reemplazo permite eliminar a la constante por completo. Para mostrar que los reemplazos son correctos se necesitan nuevas definiciones.

Definición 3 (sustitución simple)**:** es un mapeo $c \to t(x_1, \ldots, x_n)$, donde $t(\ldots)$ es un término en el que figuran los símbolos de constante $\{x_1, \ldots, x_n\}$ y «c» es un símbolo de constante aislado que no aparece en el término. Si se le da nombre, digamos σ, se escribe $\sigma : c \to t$. A la constante se la conoce como *parte izquierda* de la sustitución, o $izq(\sigma)$, y al término como *parte derecha* o $der(\sigma)$. A diferencia de las ecuaciones, las sustituciones simples están orientadas y eso facilita su uso para realizar reemplazos.

La Definición 3 está inspirada en la definición de «sustitución» que se presenta en [DP01, Klo92]. Añado el calificativo «simple» para diferenciarla de aquella pero, por legibilidad, a lo largo del trabajo omitiré el calificativo si no se presta a confusión.

Definición 4 (reemplazo)**:** es la aplicación de una sustitución simple en la reescritura de un término; si $\{t, t_1, t_2, \ldots, t_n\}$ son términos, σ la sustitución $c \to t'$, c y d son distintos símbolos de constantes y f un símbolo n-ario de función, $t\sigma$ es la imagen de t bajo σ definida del siguiente modo:

$$t\sigma = \begin{cases} t' & \text{si } t \text{ es } c \\ d & \text{si } t \text{ es } d \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{si } t \text{ es } f(t_1, \ldots, t_n). \end{cases}$$

La ampliación de la definición a la imagen de una fórmula, especificación o sistema de ecuaciones bajo una sustitución sigue una extensión homomorfa. Por su parte, dada $\sigma' : d \to t''$ con $d \neq c$, si d no aparece en t' se define $\sigma'\sigma : d \to t''\sigma$.

Sean Q un sistema de ecuaciones y $L = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ una lista de sustituciones, se define $reemplazar(L, Q)$ como la función que devuelve un nuevo sistema de ecuaciones Q' tal que $Q' = Q$ si $n = 0$, y $Q' = reemplazar(L', Q\sigma_1)$ si $n > 0$ y existe $L' = [\sigma_2\sigma_1, \ldots, \sigma_n\sigma_1]$.

Definición 5 ($E_=$)**:** dado E una especificación, $E_=$ es el sistema de ecuaciones formado por el conjunto de fórmulas de E.

Definición 6 (f/E-sustitución)**:** dada $f : c = t(x_1, \ldots, x_n)$ una f.s., se llama sustitución simple que remite a f, o *f-sustitución*, a $\sigma_f : c \rightarrow t(x_1, \ldots, x_n)$. Cuando la fórmula esté sobreentendida puede omitirse el subíndice. Notar que puede haber hasta dos sustituciones σ_f y σ'_f remitiendo a una misma f.s. f, si ambos miembros de la ecuación son constantes aisladas distintas. Por extensión, si σ es f-sustitución de f en el sistema de ecuaciones Q, σ también se dice una *Q-sustitución* y, si σ es $E_=$-sustitución de la especificación E, por comodidad también se le dice E-sustitución.

Definición 7 ($E_\rightarrow$)**:** dado Q un sistema de ecuaciones, $Q_\rightarrow$ es el conjunto de todas las Q-sustituciones. Por extensión, si E es una especificación, se nota $E_\rightarrow$ para indicar $E_{=\rightarrow}$.

Definición 8 (modelo)**:** siguiendo [DP01], una estructura $\mathcal{M}$ se dice *modelo* del sistema de ecuaciones Q, escrito $\mathcal{M} \models Q$, si $\mathcal{M}$ satisface cada fórmula de Q. Si Q y Q' son sistemas de ecuaciones, se escribe $Q \models Q'$ si todas las estructuras $\mathcal{M}$ que satisfacen Q también satisfacen Q'. Si $Q \models Q'$ y $Q' \models Q$ se escribe $Q \equiv Q'$.

Teorema 1: sea E una especificación, y σ_f una E-sustitución, $E_=\sigma_f \equiv E_=$.

Demostración: considero los casos: 1) $E_=\sigma_f \models E_=$ y 2) $E_= \models E_=\sigma_f$.

1) Caso $E_=\sigma_f \models E_=$

Sean $f : x = t(x_1, \ldots, x_n) \in E_=$ y $\mathcal{M}$ tal que $\mathcal{M} \models E_=\sigma_f$. Si $\alpha \in E_=\sigma_f$ contiene el término $t(x_1, \ldots, x_n)$, como $\mathcal{M} \models \alpha$ se tiene que, si $\mathcal{M} \models f$ entonces $\mathcal{M} \models \alpha|^x_{t(x_1,\ldots,x_n)}$; luego, $\mathcal{M} \models E_=$.

2) Caso $E_= \models E_=\sigma_f$

Sea $\mathcal{M}$ tal que $\mathcal{M} \models E_=$, en particular $\mathcal{M} \models f$ para $f : x = t(x_1, \ldots, x_n)$ en $E_=$. Sea α tal que $\mathcal{M} \models \alpha$, se tiene que $\mathcal{M} \models \alpha|^{t(x_1,\ldots,x_n)}_x$. Finalmente, como $\mathcal{M} \models E_=$ entonces $\mathcal{M} \models E_=|^{t(x_1,\ldots,x_n)}_x$ y consecuentemente $\mathcal{M} \models E_=\sigma_f$. □

Si se recorre la especificación y, cada vez que se encuentra una f.s., se realizaran inmediatamente los reemplazos posibles, el trabajo a efectuar sería muy fácil pero se perdería una propiedad atractiva. Se quisiera lograr el mejor reemplazo de constantes, es decir, el que consiga la especificación de menor complejidad final, y encontraríamos obstáculos si siguiéramos esa estrategia.

Supongamos las ecuaciones $a = f(b)$ y $b = g(a)$ con a y b símbolos distintos de constantes. ¿Qué reemplazo conviene efectuar? Debería resultar sencillo advertir que ambos no pueden eliminarse a la vez; en principio quisiéramos quitar al más pesado pero, como se verá más tarde, este problema en general no es fácil de resolver. La respuesta debe contemplar que al efectuar un reemplazo se están descartando soluciones, potencialmente mejores que las que estamos considerando, a menos que se tomen precauciones.

Una forma conservadora de preservar las soluciones óptimas consiste en evaluar ordenadamente todas las combinaciones de sustituciones y recordar cuál fue la que mejor resultado consiguió. En este caso, dada una especificación E, la solución a nuestro problema será una lista de E-sustituciones $S = [\sigma_1, \ldots, \sigma_n]$. Para desechar los casos en que una solución propone reemplazos inviables, se pide además que la lista sea *compatible*.

Antes de formalizar este concepto hay que advertir que su definición esconde una descripción informalmente sencilla: en la lista no podrá haber dos sustituciones con una misma parte izquierda ni sustituciones recíprocas entre símbolos de constantes. Para dar un ejemplo será conveniente disponer de más precisión.

Definición 9 (lista compatible)**:** dado Q un sistema de ecuaciones, una lista $L = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ de Q-sustituciones simples se dice *compatible* si es vacía o $L' = [\sigma_2\sigma_1, \sigma_3\sigma_1, \ldots, \sigma_n\sigma_1]$ es una lista compatible de $Q\sigma_1$-sustituciones; a una lista S se le dice Q-*compatible* si sus elementos son Q-sustituciones simples y, además, es compatible. Por comodidad, si E es una especificación y L es $E_=$-compatible, a L se le dice también E-compatible.

Por ejemplo, dado $Q = \{f_1 : a = f(b), f_2 : b = g(a), f_3 : b = h(c)\}$ un sistema de ecuaciones, las listas $L_1 = [\sigma_{f_1}, \sigma_{f_2}]$ y $L_2 = [\sigma_{f_2}, \sigma_{f_3}]$ no podrán ser solución del problema pues serán incompatibles. Ni $\sigma_{f_2}\sigma_{f_1}$ ni $\sigma_{f_3}\sigma_{f_2}$ son sustituciones y, entonces, ni L'_1 ni L'_2 son listas de $E\sigma_{f_1}$-sustituciones y $E\sigma_{f_2}$-sustituciones, respectivamente; en el primer caso, el símbolo de constante «b» figuraría en ambas partes de la sustitución; en el segundo, la parte izquierda no estaría conformada por un símbolo de constante aislado.

Para sintetizar informalmente, si se tiene una especificación E se desea una lista E-compatible S tal que, mediante reemplazos sucesivos, se obtenga otra especificación de complejidad mínima.

Definición 10 (objetivo)**:** sean E una especificación y «lc» una función que a partir de un conjunto de sustituciones devuelve el conjunto de todas las listas compatibles que se pueden construir con los elementos recibidos, el *objetivo* de este trabajo es encontrar S una lista de E-sustituciones compatibles tal que, si $E' = reemplazar(S, E)$, entonces

$$complejidad(E') = \min_{L \in lc(E_{\rightarrow})} complejidad(reemplazar(L, E)).$$

2.2. Planteo abstracto

Una vez presentado el problema concreto se puede idear su solución, e incluso llevarla a la práctica, con los elementos ya introducidos. Sin embargo, este curso obliga a repensar conceptos que ya están maduros desde hace tiempo en otras áreas del conocimiento.

Voy a adaptar el problema para aprovechar parte de lo que se sabe en teoría de grafos. Para empezar, se debe plantear el problema concreto como un problema abstracto sobre grafos y para eso se van a necesitar varias definiciones.

Definición 11 (grafos y digrafos)**:** siguiendo [Har69], un *grafo simple* es un par $G =< V, X >$, con V un conjunto finito de vértices o nodos y X un conjunto de aristas o arcos; un *arco* es un par no ordenado de nodos distintos de V. Si $(u, v) \in X$ se dice que u y v son *adyacentes*. Un *grafo dirigido* o *digrafo* es similar a un grafo simple, salvo por sus arcos que son pares ordenados. Si (u, v) es un arco dirigido se dice que v es adyacente «*desde*» u y que u es adyacente «*hacia*» v. Cuando no resulte ambiguo se podrá usar la expresión «grafo» para indicar «grafo simple» o, también, «grafo dirigido».

Definición 12 (hipergrafos no dirigidos)**:** siguiendo [GLNP93], un *hipergrafo simple* es un par $H =< V, X >$, con V un conjunto finito de vértices o nodos y

X un conjunto de (hiper)arcos; un *hiperarco* es un conjunto de nodos $A \subseteq V$. Entre nodos, la adyacencia extiende la definición en grafos simples. Notar que los grafos simples son casos particulares de hipergrafos no dirigidos; sin embargo, cuando no resulte ambiguo se podrán usar las expresiones «grafo» o «hipergrafo» para indicar «hipergrafo simple».

Definición 13 (hipergrafos dirigidos)**:** un *hipergrafo dirigido* o *hiperdigrafo* es similar a un hipergrafo simple, salvo por sus arcos que son pares ordenados $e = (O_e, D_e)$ de conjuntos disjuntos, quizá vacíos, de nodos de V; a O_e se le llama *origen* de e y a D_e *destino*. Entre nodos, la adyacencia «desde» y «hacia» extiende la definición en digrafos. Notar que los digrafos son casos particulares de hipergrafos dirigidos; sin embargo, cuando no resulte ambiguo se podrán usar las expresiones «grafo», «digrafo» o «hipergrafo» para indicar «hipergrafo dirigido».

También, los arcos de un hiperdigrafo se dicen *hacia adelante* (*forward*) si sus orígenes son conjuntos unitarios y *hacia atrás* (*backward*) si sus destinos lo son; un *hipergrafo hacia adelante* es un hipergrafo dirigido cuyos arcos son hacia adelante y un *hipergrafo hacia atrás*, con sus arcos hacia atrás. Es posible aliviar la escritura de conjuntos unitarios refiriéndose a ellos a partir de su elemento característico.

Definición 14 (tamaño)**:** dado un hipergrafo, se lo considera el cardinal del conjunto de arcos; se lo denota m si el grafo es implícito, y $tam(H)$ o m_H si es explícito.

Definición 15 (subgrafo)**:** dado un hipergrafo, es un nuevo grafo que tiene todos sus nodos y arcos en el original.

Definición 16 (grafo subyacente)**:** dado $G =< V, X >$ un grafo dirigido, se le dice *grafo no dirigido subyacente* a un nuevo grafo $G' =< V, X' >$ tal que sus arcos son pares no ordenados que se corresponden con los elementos de X. Dado $H =< V, X >$ un hipergrafo dirigido, se le dice *digrafo subyacente* a un nuevo grafo dirigido $G =< V, X' >$ tal que X' es el conjunto más pequeño donde, si $(\{o_1, \ldots, o_n\}, \{d_1, \ldots, d_m\})$ es un arco de H entonces $\{(o_1, d_1), \ldots, (o_1, d_m), \ldots, (o_n, d_m)\} \subseteq X'$.

Definición 17 (caminos y ciclos en grafos)**:** un *camino* en un grafo simple es una secuencia de nodos adyacentes; se le dice *simple* si todos sus nodos son diferentes. Un *ciclo* es un camino que empieza y termina en el mismo nodo y se le dice *simple* si tiene más de dos nodos y todos los intermedios son distintos de los extremos y entre sí.

Un *camino dirigido* es una secuencia de nodos de un grafo dirigido tal que cada elemento es adyacente desde su predecesor. Las definiciones de *camino dirigido simple*, *ciclo dirigido* y *ciclo dirigido simple* en digrafos son análogas a las del caso no dirigido.

Definición 18 (caminos y ciclos en hiperdigrafos)**:** se entenderá que[1] un camino dirigido en un hiperdigrafo es un camino dirigido en el digrafo subyacente. Se extienden las restantes definiciones de caminos y ciclos siguiendo la misma idea. Si no es ambiguo, puede decirse «camino» para indicar «camino dirigido».

[1] Ver [Fag83] para una discusión al respecto.

Definición 19 (dahg)**:** un hipergrafo se dice *acíclico* si no contiene ningún ciclo simple. A los grafos dirigidos acíclicos se los conoce también como *dag*, por sus siglas en inglés (*directed acyclic graph*), o como *dahg*, en el caso de los hiperdigrafos.

Definición 20 (grafo de sustituciones - g.s.)**:** es un hipergrafo hacia adelante cuyos arcos tienen destino no vacío y cuyos nodos aparecen todos en el origen o en el destino de algún hiperarco.

Con estas definiciones se quiere traducir el problema concreto a un planteo abstracto. El primer paso será convertir la especificación en un grafo; luego se reformularán el criterio para medir complejidad y el objetivo original, también abstractamente. Se puede decir informalmente que las sustituciones serán hiperarcos y que las constantes intervinientes serán nodos del g.s. que represente a la especificación original.

Definición 21 (representación)**:** dado Q un sistema de ecuaciones se define $rep(Q)$, la *representación* de Q, como el g.s. $H =< V, X >$ tal que:

- $v \in V$ sii v es un símbolo de constante en una f.s. de Q;
- $e = rep(\sigma) = (c, \{x_i\}_{1 \leq i \leq n}) \in X$ sii existe $\sigma : c \to t(x_1, \ldots, x_n)$ en $Q_{\to}$ y $\{c\} \cup \{x_i\}_{1 \leq i \leq n}$ son los símbolos de constantes en σ.

Dada E una especificación se escribe $rep(E)$ para indicar $rep(E_{=})$ y se dice, indistintamente, que el g.s. representa a la especificación o a su sistema de ecuaciones.

Al adaptar la especificación sucedieron varios cambios, algunos poco evidentes. Mientras que en la especificación se encontraban símbolos de constantes, variables y funciones, tanto formando parte de fórmulas simples como no, en el grafo de sustituciones sólo quedan plasmadas las constantes que figuran en fórmulas simples; como consecuencia no es posible reconstruir la especificación original exclusivamente a partir del grafo que la representa pero, a cambio, se obtiene una herramienta de trabajo más focalizada en los aspectos relevantes. Otro desprendimiento de la transformación es que, por carecer de parte de la información inicial, no puede definirse la complejidad de un g.s. del mismo modo que antes. Es necesario modificar el objetivo a perseguir.

Definición 22 (ahorro en especificaciones)**:** dadas E una especificación y L una lista E-compatible,

$$ahorro(L, E) = complejidad(E) - complejidad(reemplazar(L, E)).$$

Con E una especificación, en vez de buscar minimizar su complejidad resulta equivalente intentar maximizar el ahorro posible; equivalencia que surge de

$$complejidad_{\text{mín}}(E) = complejidad(E) - ahorro_{\text{máx}}(E)$$

para definiciones razonables de $complejidad_{\text{mín}}$ y $ahorro_{\text{máx}}$. Antes de continuar, este problema dual necesita nuevas definiciones.

Definición 23 (colección)**:** uso el término «colección» para denotar indistintamente listas o conjuntos.

Definición 24 (colección conflictiva)**:** dado H un g.s., una colección C de arcos del grafo se dice *conflictiva* si sus elementos pueden formar un ciclo en H. Un g.s. se dice él mismo *conflictivo* si su conjunto de arcos es conflictivo. Si bien esta definición no introduce ningún concepto novedoso, resulta muy útil para facilitar algunas explicaciones futuras.

Definición 25 (grado de un nodo)**:** dado $H =< V, X >$ un hipergrafo dirigido y C una colección de arcos en X, el *grado de entrada* $d^C_{in}(v)$ de un nodo $v \in V$ es la cantidad de arcos en C que tienen a v en el destino y el *grado de salida* $d^C_{out}(v)$ es la cantidad de arcos en C que tienen a v en el origen. Si no hay ambigüedad se puede omitir la colección para indicar el propio conjunto X de arcos del grafo, o bien se puede utilizar al grafo en sí como superíndice para indicar también a su propio conjunto de arcos.

Definición 26 (hoja)**:** dado un hiperdigrafo $H =< V, X >$, un nodo $v \in V$ se dice *hoja* si es tal que $d^H_{out}(v) = 0$.

Definición 27 (colección determinística)**:** dado $H =< V, X >$ un g.s., una colección C de arcos del grafo se dice *determinística* si es no-conflictiva y, para todo $v \in V$, $d^C_{out}(v) \leq 1$. Un g.s. se dice *determinístico* si su conjunto de arcos es determinístico.

Vale aclarar que las colecciones determinísticas de arcos encarnan la misma noción que las listas compatibles de sustituciones, afirmación que más adelante se demostrará, y que la decisión de atribuirles distintos nombres pretende resaltar la diferencia de dominios involucrados.

Definición 28 (ganancia)**:** dado $H =< V, X >$ un g.s.[2], una función g_H se dice *ganancia* para H si asigna un número real $g_H(v) > 1$ a cada $v \in V$; por extensión, la ganancia de $e = (o_e, D_e) \in X$ se define como $g_H(e) = g_H(o_e)$. Puede omitirse el subíndice si el grafo se sobreentiende.

Definición 29 (ahorro en grafos)**:** sean H un g.s., C una colección determinística de arcos del grafo y «g» una función ganancia para H,

$$ahorro_g(C) = \prod_{e \in C} g(e).$$

Puede omitirse el subíndice de $ahorro_g$ si la función ganancia se sobreentiende.

Las nuevas definiciones permiten volver a plantear el objetivo, pero prescindiendo de la especificación original; su lugar es ocupado por el g.s. que la representa. Luego se demostrará que esta transformación es correcta.

Definición 30 (objetivo alternativo)**:** dados $H =< V, X >$ un g.s., «g» una función ganancia para H y «lc'» una función que a partir de una colección de arcos devuelve el conjunto de todas las listas determinísticas que se pueden construir con los elementos recibidos, el *objetivo alternativo* radica en encontrar

[2]En otros problemas la definición de ganancia puede ser distinta.

una lista[3] determinística Z de arcos de H, a la que se le dirá *solución* para H, tal que

$$ahorro_g(Z) = \max_{L \in lc'(X)} ahorro_g(L).$$

Para que tenga sentido la transformación de especificaciones a grafos hay que demostrar que resolver el objetivo alternativo soluciona el problema original. Antes se necesita una definición adicional.

Definición 31 (apellido)**:** dados Q un sistema de ecuaciones y $H =< V, X >$ un g.s. tal que $H = rep(Q)$, una función ap se dice *apellido* de X en $Q_{\rightarrow}$ si, cuando $e = (c, \{x_1, \ldots, x_n\}) \in X$ y $ap(e) = \sigma_f$, entonces σ_f es $c \rightarrow t(x_1, \ldots, x_n)$.

Teorema 2: sean E una especificación, $H =< V, X >= rep(E_{=})$ un g.s., «g» una función ganancia para H tal que $g(c) = peso_E(c)$ para todo $c \in V$, $Z = [e_1, \ldots, e_n]$ una lista determinística de arcos de H que satisface el objetivo alternativo y ap una función apellido de X en $E_{\rightarrow}$, entonces la lista $S = [ap(e_1), \ldots, ap(e_n)]$ es solución de E para el objetivo original.

Demostración: de la definición del objetivo original se desprende que demostrar este teorema resulta equivalente a probar que, en las condiciones del teorema:

- S sea una lista E-compatible;
- S minimice la complejidad de E.

Asumiendo válidas ambas proposiciones es directo que S es solución de E para el objetivo original. □

Afirmación 1: en las condiciones del Teorema 2, con $Z = [e_1, \ldots, e_n]$ una lista cualquiera de arcos de H, la lista $S = [ap(e_1), \ldots, ap(e_n)]$ es E-compatible si, y sólo si, Z es determinística.

Demostración: la justificación de esta nueva afirmación abunda en detalles minuciosos y poco interesantes, razón por la cual se la puede consultar en el Apéndice A.2. De todos modos, cabe aclarar que es esperable que *compatibilidad* y *determinismo* resulten equivalentes, ya que ambas nociones fueron definidas con el preciso propósito de reflejar un mismo concepto, cada una en su dominio: las sustituciones o los grafos según corresponda. □

Corolario: en las condiciones del Teorema 2, de la Afirmación 1 se sigue que S es una lista E-compatible. □

Afirmación 2: en las condiciones del Teorema 2, S minimiza la complejidad de E.

Demostración: con E una especificación, $H =< V, X >= rep(E)$, «g» una función ganancia tal que $g(c) = peso_E(c)$ para todo $c \in V$, $Z = [e_1, \ldots, e_n]$ una lista determinística de arcos de H que satisface el objetivo alternativo, ap una función apellido de X en $E_{\rightarrow}$ y $S = [ap(e_1), \ldots, ap(e_n)]$, voy a argumentar por el absurdo.

[3]Más adelante se verá que es conveniente buscar un conjunto y no una lista, pero por el momento la lista facilita la comparación con las especificaciones.

Si se supone que S no minimiza la complejidad o, lo que es lo mismo, no maximiza el ahorro para E, debe existir $M = [\sigma_1, \ldots, \sigma_m]$ E-compatible tal que

$$ahorro(M, E) > ahorro(S, E).$$

Pero, como por suposición $\sigma_i \in E_{\rightarrow}$ para $1 \leq i \leq m$, existen e'_i arcos en $rep(E)$ tales que $e'_i = rep(\sigma_i)$; entonces $R = [e'_1, \ldots, e'_m]$ existe y, por la Afirmación 1, es determinística. Y como era un requisito que $g(c) = peso_E(c)$, también se tiene que

$$\begin{array}{rcl} ahorro(S, E) & = & ahorro_g(Z) \quad \text{y} \\ ahorro(M, E) & = & ahorro_g(R). \end{array}$$

Lo que resulta una contradicción: por hipótesis, $ahorro_g(Z)$ es máximo pero, de lo anterior, se desprende que

$$ahorro_g(R) > ahorro_g(Z).$$

Como el absurdo provino de suponer la existencia de la lista M, se concluye que S minimiza la complejidad de E. □

En este punto ya se vio que el objetivo radica, en el fondo, en enfrentar un problema de optimización. Dado un g.s., en el espacio de búsqueda de todas las colecciones determinísticas de sus arcos se quiere aquella que maximice la función objetivo *ahorro* o, alternativamente, se puede pensar que se está buscando un subgrafo determinístico óptimo. Ya que el dominio de la función objetivo es discreto, además, se puede precisar que se está enfrentando un problema de optimización combinatoria.

Hasta aquí se presentaron explicaciones y definiciones pero no se dieron muchos indicios de cómo se piensa resolver el problema. Se quisiera disponer de un algoritmo rápido y eficiente que garantice una solución globalmente óptima. ¿Será ésto posible? Lamentablemente, a menos que $P = NP$, la respuesta es negativa.

2.3. Complejidad

Para justificar la complejidad temporal del problema voy a presentar una idea[4] que no es mía. Se trata de mostrar que de la solución a este problema deriva la respuesta a otro conocido problema *NP-completo*; luego, salvo que $P = NP$, el que se quiere resolver no tendrá solución polinomial. Comienzo adaptando el objetivo alternativo a un problema de decisión.

Problema: SubHipergrafo Acíclico Óptimo (SHAO)

Instancia (I_{SHAO})**:** un g.s. $H =< V, X >$, una función ganancia g sobre V y un número real c.

Pregunta: ¿existe una colección determinística C de arcos de H tal que $ahorro_g(C) \geq c$?

Ahora el otro...

[4]Vale un *gracias* a Flavia Bonomo por la ayuda y la paciencia.

Problema: Nodos de realimentación – Feedback Vertex Set (FVS)

Instancia (I_{FVS})**:** un grafo dirigido $G =< V, X >$ y un entero positivo k.

Pregunta: ¿existe un subconjunto $V' \subseteq V$ con $|V'| \leq k$ tal que V' contiene al menos un nodo de cada ciclo dirigido de G?

Se sabe por [GJ79] que en general FVS es NP-completo.

Teorema 3: SHAO es NP-completo.

Demostración: Dado que SHAO está en NP describo cómo construir I_{SHAO} a partir de una instancia de FVS; se tiene I_{FVS} como el digrafo $G =< V, X >$ y un entero positivo k.

Se definen:

- el mínimo conjunto X' de arcos de un g.s. de modo que, si $v, u_1, \ldots, u_n$ son nodos de V y $\{(v, u_1), \ldots, (v, u_n)\}$ son todos los arcos dirigidos de X con origen en v, entonces $(v, \{u_1, \ldots, u_n\}) \in X'$. Con este conjunto se determina el g.s. $H =< V, X' >$. Notar que si $v \in V$, $d^H_{out}(v) \leq 1$.
- la función constante $g : V \to \mathbb{R}$ de modo que, si $v \in V$, $g(v) = 2$.
- el número real $c = 2^{|X'|-k}$.

Sea I_{SHAO} el g.s. H con la función ganancia g y el número c, se obtiene que la respuesta a I_{SHAO} debe ser la misma que la de I_{FVS}. Veamos por qué.

Si la respuesta a I_{FVS} es *sí* significa que existe $V' \subseteq V$ con $|V'| \leq k$ tal que V' contiene al menos un nodo de cada ciclo dirigido de G. Entonces, si se define

$$C = \{(o, D) \in X' / o \notin V'\}$$

se obtiene un subconjunto de arcos de X' al que se le quita al menos un arco de cada ciclo; es decir, una colección no conflictiva. Y como $d^H_{out}(v) \leq 1$ para $v \in V$, además C es determinística.

Con C y g se consigue $ahorro(C) = 2^{|C|}$, para lo cuál interesa que $|C| = |X'| - |V'|$ ya que a los elementos de X' se les quita exactamente un arco para cada nodo en V'.

Hay que determinar si es cierto que

$$ahorro(C) \geq c$$

sabiendo que $ahorro(C) = 2^{|X'|-|V'|}$ y $c = 2^{|X'|-k}$, lo que es directo pues $|V'| \leq k$ es cierto por hipótesis y por tanto vuelve verdadera a la afirmación.

Se tiene que si la respuesta a I_{FVS} es *sí*, también deberá serlo para I_{SHAO}.

Si la respuesta a I_{SHAO} es *sí* significa que existe una colección determinística C de arcos de H tal que $ahorro(C) \geq c$. Entonces, si siendo $A \subseteq X'$ anoto $\tilde{A} = \{v \in V / \exists D : (v, D) \in A\}$, y se define

$$V' = \tilde{X'} - \tilde{C}$$

se obtiene un importante subconjunto de V.

Para ver que V' contiene un nodo de cada ciclo de G alcanza con advertir que como $\tilde{C} \subseteq \tilde{X}'$ y $\tilde{C} \cup V' = \tilde{X}'$ vale que $\tilde{C} = \tilde{X}' - V'$ y, si no hubiera en V' un nodo de cada ciclo, en C podría haber ciclos y por hipótesis no los hay.

Con $\tilde{C} \subseteq \tilde{X}'$ y la definición de V' se consigue $|V'| = |\tilde{X}'| - |\tilde{C}|$, para lo cuál interesa que $|\tilde{X}'| = |X'|$ y $|\tilde{C}| = |C|$ ya que $d^H_{out}(v) = 1$ para todo $v \in \tilde{X}'$.

Hay que determinar si es cierto que

$$|V'| \leq k$$

sabiendo que $2^{|C|} = ahorro(C) \geq c = 2^{|X'|-k}$ y, de lo anterior, $|C| = |X'| - |V'|$, lo que es directo pues $|X'| - |V'| \geq |X'| - k$ es cierto por hipótesis y por lo tanto vuelve verdadera a la afirmación.

Se tiene que si la respuesta a I_{SHAO} es *sí*, también deberá serlo para I_{FVS}.

Dado que valen razonamientos iguales para un *no* como respuesta, FVS es NP-completo y la reducción de una instancia de FVS en una de SHAO es polinomial, se implica que SHAO es también NP-completo. □

2.4. Algoritmo general

Enfrentar un problema NP-completo inclina el ánimo a favor de dejar a un lado la esperanza de conseguir simultáneamente una solución general, rápida y óptima globalmente; habrá que encontrar algún compromiso.

Para acomodar distintas estrategias de solución, a continuación presento un esqueleto general y algunas nociones complementarias, comunes todos a una gran variedad de posibilidades. Las estrategias concretas y los análisis de complejidad temporal quedan postergados hasta que se hayan introducido todos los conceptos relevantes.

2.4.1. Esqueleto

Dado que probablemente no sea posible conseguir una única estrategia satisfactoria para resolver todas las instancias interesantes, una alternativa viable será disponer de varias, cada una con sus ventajas y limitaciones, para elegir según la ocasión. Con el fin de facilitar esta multiplicidad conviene distinguir entre aquellos pasos que resultan atractivos de modificar para cada estrategia y aquellos que se desea que permanezcan invariantes.

Mientras más pasos en común haya entre las alternativas, más sencilla resultará la comparación entre ellas pero menor será el grado de libertad con que contarán. En particular, será preferible dejar a consideración de las estrategias solamente los detalles que consigan una diferencia sustantiva y, con ese espíritu, plasmar en un esqueleto compartido la mayor cantidad posible de elementos comunes. Juntos, este esqueleto y las estrategias que se presentan más tarde son suficiente para buscar soluciones.

El esqueleto propone un marco que todas las variantes deben respetar pero que, a la vez, permite la manifestación de distintos comportamientos dependiendo de la estrategia vigente. En la implementación el esqueleto aparece bajo el nombre `optimize(...)`, a raíz de la tarea genérica que desempeña; su pseudocódigo se presenta en la Figura 1.

```
optimize(Collection<Candidate> candidates)
{
  if candidates.isEmpty()
    if solution = null or solutionSaving < partialSolutionSaving
      solution <- partialSolution;
  else {
    selectedCandidates <- strategy.selectCandidates(candidates);
    for_each Candidate c in selectedCandidates {
      partialSolution.add(c);
      optimize(updateCandidates(c, candidates));
      partialSolution.remove(c);
    }
  }
}
```

Figura 1: Esqueleto

Esquemáticamente, el esqueleto construye soluciones y evalúa su calidad para quedarse con una de las mejores. La porción más importante es la que delega en alguna estrategia el criterio de selección de candidatos; en la implementación, ésto se corresponde con la invocación a `selectCandidates(...)` para la estrategia corriente. El objeto `strategy` abstrae las alternativas. Tras esa llamada se actualiza el universo de elementos conforme a la selección y se analiza el problema resultante, de menor complejidad. Si durante la selección el criterio elige varios candidatos en un único paso, el esqueleto los analizará consecutivamente, como si se tratara de múltiples casos individuales aislados, sin considerarlos en simultáneo dado que podrían ser conflictivos entre sí.

Si resulta llamativo advertir la mención de una «colección» (`Collection`) de elementos, vale recordar que este término es intencional por cuanto pretende evitar precisar si se trata de una lista o un conjunto. Pronto se atenderá esta imprecisión.

Dos nociones centrales al funcionamiento de este algoritmo son las de *solución* y *solución parcial*. La *solución* representa a la mejor combinación de candidatos encontrada hasta el momento, mientras que la *solución parcial* a la construcción corriente, tal vez incompleta, de una nueva solución factible. Durante la ejecución constantemente se modifica la solución parcial agregándosele un elemento después de cada selección y removiendo, ese mismo elemento, antes de analizar al siguiente de una selección múltiple. La mejor solución conocida, en cambio, sólo se actualiza en el caso base y a condición de que la nueva solución encontrada supere en calidad a la mejor conocida hasta entonces. `solutionSaving` y `partialSolutionSaving` representan el ahorro de la mejor solución conocida y de la solución parcial en curso, respectivamente, ambos calculados según la Definición 29.

El contexto simplificado en que el esqueleto de optimización ejecuta se completa con el programa principal `main(...)` que lo invoca e inicializa sus variables externas; el pseudocódigo se ve en la Figura 2. El programa principal recibe del usuario la especificación cuya complejidad se quiere minimizar, inicializa en *null* la mejor solución conocida para indicar que todavía está indefinido este valor, construye una solución parcial de trabajo e invoca a la optimización después de

extraer de la especificación los candidatos a sustituciones; por último almacena una nueva especificación simplificada, producto de aplicar a la original las sustituciones de la mejor solución que pudo encontrarse.

```
main(Spec spec)
{
  solution <- null;
  partialSolution <- new Collection<Candidate>();
  optimize(extractCandidates(spec));
  write(apply(solution, spec));
}
```

Figura 2: Programa principal

Ahora es posible remitir tanto el problema concreto como su planteo abstracto a las porciones de implementación vistas. El programa principal recibe explícitamente una especificación pero el algoritmo de optimización, en cambio, recibe una colección de candidatos a sustitución, extraídos de la especificación. En el comienzo, la especificación incluye un conjunto de fórmulas del cual se extraen las *fórmulas simples* (Definición 2) que se usarán para minimizar la complejidad de la especificación; entonces, `extractCandidates(spec)` recolecta fórmulas simples (f.s.) pero las transforma en *candidatos* a sustitución.

La clave de la transformación de f.s. en candidatos a sustitución se halla en la Definición 21 (representación) del planteo abstracto. Para una primera aproximación alcanza con entender a los candidatos a sustitución como elementos capaces de mejorar una solución parcial, constituidos por pares ordenados (σ_f, e) en los que σ_f es una sustitución que remite a f, una f.s. en la especificación, mientras que «e» es el arco de un grafo de sustituciones (g.s.), se inicializa en $rep(\sigma_f)$ y almacena toda la información necesaria para buscar soluciones; la relación entre ambos nace de que, de alguna manera, la sustitución implementa para el arco el concepto de la Definición 31 (apellido), necesario para efectuar reescrituras en la especificación a partir de la mejor solución encontrada para la representación sobre grafos. En el Apéndice A.1 se retoma esta explicación más profundamente.

A diferencia de los arcos, los nodos del g.s. no figuran explícitamente en el algoritmo sino que se infieren a partir de los orígenes y destinos de los arcos. El grafo del planteo abstracto es construido indirectamente, antes de comenzar la optimización, como hiperarcos dentro de la colección de candidatos.

Definido el esqueleto y su contexto es necesario dedicar un momento a justificar que el algoritmo sea correcto. Interesa garantizar la terminación para cualquier entrada. Para ello hay que tener en cuenta que la especificación contiene un conjunto finito de fórmulas, de las cuales sólo algunas se transformarán en candidatos a sustitución; luego, el esqueleto de optimización recibirá necesariamente una colección finita de candidatos. Si esa colección no es vacía se exige a las estrategias que su selección de candidatos tampoco lo sea; además, se exige a la actualización del universo que asegure disminuir la cantidad de elementos de la colección. Bajo estas condiciones, una cantidad finita, discreta y no negativa de elementos que siempre disminuye da garantía de que el algoritmo siempre termina.

A la par de asegurar terminación, para justificar que el algoritmo sea correcto también se debe definir qué se espera que compute, para luego comprobar que siempre se comporta como se espera. Del esqueleto se espera que construya soluciones conforme al criterio de cada estrategia y que, en caso de construir más de una, preserve alguna que consiga ahorro máximo. Que el algoritmo construya soluciones conforme al criterio vigente resulta de su propia construcción. Cada vez que hay candidatos para elegir, es la estrategia quien selecciona sustituciones. El esqueleto se encarga de añadir a la solución parcial del momento cada uno de los elegidos, consecutivamente. En el caso base se observa que, cuando el algoritmo encuentra una solución, compara la ganancia de la recién encontrada contra la de la mejor conocida; basándose en la comparación, cada vez se preserva a la de mayor ahorro.

Por último, en lo que respecta al comportamiento correcto del algoritmo se espera que las soluciones encontradas sean viables y, de nuevo, será la actualización del universo de candidatos la encargada de esta garantía. Por el momento conviene posponer las explicaciones pendientes, pero igual puede aclararse que al seleccionar un candidato la actualización devuelve una colección de elementos que, por definición, conducen a soluciones viables sin importar la estrategia.

2.4.2. Colecciones y permutaciones

Desde el comienzo se está evitando precisar si las colecciones involucradas se refieren a listas o conjuntos de elementos. La razón para postergar esta aclaración surge de la existencia de buenos argumentos en favor de cada alternativa y de la validez de ambas.

Teorema 4 (orden indistinto)**:** sean H un g.s., g una función ganancia para H, L una lista determinística de arcos de H y L' una permutación de L,

$$ahorro_g(L) = ahorro_g(L').$$

Demostración: inmediata a partir de la Definición 29 (ahorro) y la conmutatividad del producto, siempre y cuando L' sea una lista determinística. □

Teorema 5 (permutar preserva determinismo)**:** sean L una lista determinística de arcos de algún g.s. y L' una permutación de los elementos de L, también L' es determinística.

Demostración: también inmediata, esta vez a partir de la Definición 27 (colección determinística), dado que el grado de salida de un nodo con respecto a la colección no cambia si se permutan elementos, como tampoco su capacidad para formar ciclos. □

Vista la equivalencia teórica, presento las diferencias prácticas que surgen. Si se privilegia la visión como lista se resalta la naturaleza procedural que cobra la implementación y se respetan las descripciones del problema concreto y del planteo abstracto. Si se privilegia al conjunto, el tiempo de ejecución del algoritmo se reduce drásticamente (por eludir soluciones equivalentes), se evita atender a un orden artificial y se asemeja el problema a otros relacionados, como FVS, facilitando el estudio y la comparación de resultados.

¿Entonces? Pues conviene considerar la mejor forma de entender el problema según la ocasión. Durante la presentación del problema concreto, el respeto por

el orden de los elementos ayudó a explicar que se deben tener en cuenta todas las combinaciones viables de candidatos; en cuanto al planteo abstracto y el análisis de complejidad, la perspectiva sin orden se ajusta mejor. Eliminada la necesidad de respetar el orden entre elementos, el uso de listas facilita la implementación y, sin embargo, permite ignorar soluciones equivalentes si se programa con un mínimo de cuidado.

2.4.3. Actualización

Una porción crítica del esqueleto merece ser estudiada más profundamente; se trata de la actualización del universo de candidatos. En el pseudocódigo presentado dice `optimize(updateCandidates(c, candidates))` y hasta ahora sobre `updateCandidates(...)` sólo se hicieron explícitos algunos requerimientos superficiales. Comenzaré por describir más acabadamente la tarea que desempeña y analizar el algoritmo que la implementa; después muestro cómo cumple con las exigencias que se le impusieron.

Del esqueleto se desprende que al seleccionar candidatos deben suceder modificaciones en el universo; la razón remite al problema concreto. Después de que la estrategia haya elegido sus candidatos, se evaluarán las consecuencias de efectuar el reemplazo propuesto por cada uno de ellos, individual y consecutivamente. Entonces, la función de actualización reconfigura el universo como si hubiera sido reescrita la especificación original siguiendo una sustitución. Como el universo de candidatos refleja el g.s. asociado a una especificación que se modifica, también él debe adaptarse a los cambios.

Al seleccionar un candidato se quisiera aplicar al conjunto completo de fórmulas de la especificación la sustitución asociada. De realizarse este reemplazo se encontraría que, en cada actualización, un símbolo de constante desaparece efectivamente de la especificación y, al mismo tiempo, algunas fórmulas simples dejan de serlo, mientras que otras pueden sufrir modificaciones.

Concretamente, tras la selección de un candidato la tarea de actualización consiste en restaurar el universo a un estado válido. A los nuevos elementos se les pide que reflejen la aplicación de la sustitución elegida y respeten la posibilidad de incorporarse en una futura solución parcial.

Recapitulando un poco, se dijo que el problema concreto fue transformado por una razón: facilitar su estudio y tratamiento. En lo que respecta a su tratamiento se convirtieron las f.s. en candidatos a sustitución, dentro de los cuales lo que ahora importa son los arcos del g.s. implícito. En este punto los candidatos son sólo una sustitución más un hiperarco, donde la primera es inmutable, remite siempre a una misma f.s. de la especificación original y no cumple función aparente. La tarea de actualización del universo, entonces, se reduce a observar los cambios que sufren estos hiperarcos; por fuera de ellos el único cuidado restante es que, si se elimina un arco del universo, debe desaparecer también el candidato que lo alberga.

Si $s = (\sigma, e)$ es el candidato seleccionado, $c = (\sigma_c, e_c)$ es un elemento del universo y tanto $e = (v, X)$ como $e_c = (u, Y)$ son hiperarcos apropiados, para que los cambios imiten el efecto de aplicar la sustitución σ a toda la especificación, intuitivamente se quisiera reemplazar cada aparición de v por X en e_c. Para un resultado válido, las posibilidades de e_c son:

Caso 1 - $v = u$**:** tiene igual origen que e, incluso podría ser e, y como el reemplazo no está bien definido en un g.s. se lo elimina del universo.

Caso 2 - $v \neq u$ **y** $v \notin Y$**:** sus nodos no tienen intersección con los de e y los reemplazos inducidos no lo afectan; por lo tanto permanece en el universo sin cambios.

Caso 3 - $v \neq u$**,** $v \in Y$ **y** $u \in X$**:** si se reemplazara v por X quedaría u tanto en el origen como en el destino, no conformando un arco válido; se lo elimina del universo.

Caso 4 - $v \neq u$**,** $v \in Y$ **y** $u \notin X$**:** se construye $Y' = (Y - \{v\}) \cup X$, un nuevo conjunto que reemplaza a Y.

Con la descripción de la tarea a realizar, más los casos a considerar, un algoritmo que la ejecute para todos los elementos del universo se presenta en la Figura 3.

```
Collection<Candidate>
updateCandidates(Candidate s, Collection<Candidate> candidates)
{
  (v, X) <- s.arc;
  for_each Candidate c in candidates {
    (u, Y) <- c.arc;
    if u = v
      candidates.delete(c);
    elseIf v in Y
      if u in X
        candidates.delete(c);
      else
        Y <- (Y-{v})+X;
  }
  return candidates;
}
```

Figura 3: Actualización de los candidatos.

Es fácil ver que, por construcción, el algoritmo sigue los lineamientos del análisis por casos. Entonces, ya que ninguno de los elementos eliminados podría conducir a nuevas soluciones y sólo ellos son eliminados, la restauración del universo a un estado válido conserva a tantos candidatos como sea posible. Se justifica la terminación para toda entrada observando que los elementos del universo conforman una cantidad limitada, se analizan uno tras otro sin repeticiones y a cada uno se le dedica un esfuerzo acotado. Otros detalles se presentan en los Apéndices A.1 y A.3.

Resta justificar que se cumple el requerimiento de disminuir la cantidad de candidatos que se está procesando. Si se recuerda que, por contexto, el elemento seleccionado siempre forma parte del universo se ve que al menos él será necesariamente removido de la colección, ya que su parte izquierda obligadamente coincide con sí misma. Dado que ningún paso agrega elementos nuevos y al menos uno siempre es removido, se logra cumplir con el requerimiento.

2.5. Estrategias

A continuación se presentan y estudian distintas estrategias de solución: fuerza bruta, más pesado primero y GRASP. En cada caso se describe la estrategia, sus supuestos, algoritmos, validez y complejidad temporal. En el marco que establece el esqueleto anterior son posibles otras estrategias más allá de las presentes; éstas fueron elegidas como representativas de los enfoques exacto, goloso y heurístico pero otras consideraciones abren la puerta a futuras alternativas.

Es conveniente recordar que el tamaño de la entrada considerado en el estudio de complejidad está dado por el tamaño del g.s. implícito que, según la Definición 14, es la cantidad m de hiperarcos del grafo o, lo que es igual, la cantidad de elementos del universo de candidatos. Dada una instancia considero la complejidad necesaria para resolver el problema con el mismo criterio que en la presentación del problema concreto: analizo la cantidad de combinaciones a evaluar, aún sabiendo que otros factores también influyen en el esfuerzo total insumido. El factor omitido más importante está dado por las sucesivas actualizaciones del universo, que se realizan de idéntico modo para todas las estrategias y por tanto no aportan a la comparación entre estrategias.

Por último, vale aclarar que por contexto todas las estrategias presentes y futuras asumen la precondición de recibir conjuntos no vacíos de candidatos con los cuales operar y, también, la postcondición de devolver siempre selecciones igualmente no vacías.

2.5.1. Fuerza bruta

Esta estrategia consiste en seleccionar de la colección de candidatos, en cada paso, a todos los elementos o, lo que es lo mismo, no eliminar ni preferir a ninguno en particular. En el esqueleto, esto equivale a enumerar todas las posibles soluciones. El tratamiento exhaustivo garantiza una solución globalmente óptima y se presenta en la Figura 4.

```
Collection<Candidate>
Brute.selectCandidates(Collection<Candidate> candidates)
{
  return candidates;
}
```

Figura 4: Selección exhaustiva.

La validez del método resulta de su propia construcción, pues analizar todas las alternativas consigue la mejor solución posible pero insume un esfuerzo tan grande que su utilidad queda circunscrita a casos muy pequeños y, por lo tanto, no resulta satisfactorio para el caso general, que podría ser tan grande como uno quisiera. Si la entrada es no vacía resulta inmediato que la salida contendrá algún elemento.

La complejidad de peor caso es $2^{m-1} \in O(2^m)$ ya que, si se tiene un ciclo, para evitar soluciones no determinísticas al menos un candidato deberá ser omitido y la cantidad máxima de subconjuntos que pueden formar los restantes candidatos es exactamente esa; al no restringir la selección de candidatos de ninguna manera, todo este universo deberá considerarse. El caso sin ciclos se considera aparte, más adelante, y no modifica la explicación.

2.5.2. Más pesado primero

Esta estrategia consiste en seleccionar de la colección de candidatos, en cada paso, a un único elemento que aporte un ahorro individual máximo, valiéndose de la equivalencia entre ahorro y peso (*weight* en inglés). En el esqueleto, esto equivale a tomar rápidamente una decisión miope o golosa, con información parcial, tratamiento que se presenta en la Figura 5 y no garantiza una solución globalmente óptima.

```
Collection<Candidate>
HeaviestFirst.selectCandidates(Collection<Candidate> candidates)
{
  Candidate selected <- null;
  for_each Candidate c in candidates
    if selected = null or selected.weight < c.weight
      selected <- c;
  return Collections.singleton(selected);
}
```

Figura 5: Selección golosa.

La validez del método resulta de la relajación que conlleva no perseguir una solución óptima y de la factibilidad de cualquier combinación de candidatos, mediada por las actualizaciones del universo. Si la entrada es no vacía resulta claro que la salida contendrá algún elemento. A pesar de sus limitaciones, el método evita el problema del esfuerzo necesario por fuerza bruta y permite el análisis de especificaciones que de otras formas resultarían inabordables. Conviene observar que la situación inicial en cualquier caso mejora, dado que la complejidad de la especificación original siempre decrece; la contracara es que con un poco de ingenio se pueden construir instancias cuyas soluciones resulten arbitrariamente malas.

La complejidad de peor caso es $O(1)$. Se elige un único candidato en cada oportunidad hasta vaciar el universo; llegado a este punto se encontró una solución, que es la única evaluada, y se finaliza la ejecución. Si bien esta complejidad puede no resultar intuitiva, es un desprendimiento del criterio elegido, que ignora las comparaciones entre candidatos. Otros criterios son posibles y se discuten más adelante.

2.5.3. GRASP

Esta estrategia es una de las varias metaheurísticas que usualmente se utilizan para resolver problemas complejos de optimización combinatoria; otras como *simulated annealing* [KCDGV83], *tabu search* [Glo86, Glo89] o *genetic programming* [Koz95] también podrían haberse considerado. Arbitrariamente se eligió GRASP [FR89, FR95], implementado según los lineamientos propuestos en [Res99], incluida la modificación para asegurar convergencia asintótica global.

Sintetizadamente consiste en formar a partir del total de candidatos, en cada paso del algoritmo, un conjunto reducido de *buenos candidatos* en algún sentido, para luego seleccionar uno al azar de entre ellos; una vez encontrada una solución comienza una etapa de búsqueda local que intenta mejorar el resultado recién obtenido. El proceso se repite alguna cantidad de iteraciones y el resultado final

será la mejor combinación lograda en alguna corrida. Aunque se esperan soluciones de buena calidad sorteando las limitaciones de estrategias golosas, tampoco garantiza una solución globalmente óptima en una cantidad finita de iteraciones. Sin embargo, la convergencia asintótica global [Res99, RPP00] permite esperar soluciones de buena calidad si la cantidad de iteraciones es suficiente.

Respecto del comportamiento, si bien la estrategia sigue la prescripción general dictada por el esqueleto, en la práctica hay otros aspectos más a los que prestar atención. Cabe observar que el procedimiento de optimización no se ejecutará una única vez, sino tantas como iteraciones indique la configuración de la estrategia, mediante un ciclo que encierra al algoritmo de optimización y se evitó exhibir en la Figura 2. Lo único importante que resta mencionar ahora es que la implementación de este trabajo estima cuántos candidatos es capaz de estudiar en poco tiempo y considera que el problema está resuelto ni bien reduce la instancia al tamaño que considera tratable; a partir de entonces, la búsqueda local realiza un análisis exhaustivo. El Apéndice C ahonda en algunos pormenores.

Específicamente, la implementación se presenta en la Figura 6, donde se ve el computo del umbral que caracteriza a la lista restringida de candidatos. El valor real $\alpha \in [0; 1]$ controla el umbral, regula la convergencia asintótica y es elegido al comienzo de cada iteración también aleatoriamente.

```
Collection<Candidate>
GRASP.selectCandidates(Collection<Candidate> candidates)
{
  double min <- null, max <- null;

  for_each Candidate c in candidates {
    if min = null or min > c.weight
      min <- c.weight;
    if max = null or max < c.weight
      max <- c.weight;
  }

  double bound <- min + alpha * (max - min);

  Collection<Candidate> restricted <- new Collection<Candidate>()
  for_each Candidate c in candidates
    if bound <= c.weight
      restricted.add(c);

  return Collections.singleton(restricted.randomGet());
}
```

Figura 6: Selección heurística.

Con un mínimo cuidado en la programación se logra evitar recorrer los elementos para obtener máximos y mínimos, como también la construcción transitoria de la lista restringida de candidatos. El pseudocódigo es ilustrativo de la tarea que desempeña la implementación aunque no exactamente del modo en que la lleva a cabo. A pesar de que algunos detalles se posterguen vale la

aclaración para justificar que, aunque parece trabajosa, la implementación se desempeña muy eficientemente.

Al igual que en el caso de *más pesado primero*, la validez del método surge de no asegurar una solución globalmente óptima en finitas iteraciones y de la factibilidad de los resultados. Si la entrada es no vacía, para todo posible α la colección restringida contendrá algún elemento y la selección aleatoria devolverá un resultado no vacío.

El método proporciona una gama de oportunidades intermedias entre la velocidad del algoritmo goloso y la calidad de la solución de fuerza bruta. Los parámetros que ofrece para guiar su comportamiento son la cantidad de iteraciones que realizará y el tiempo durante el cual se le permite a la etapa de búsqueda local intentar mejorar una solución parcial. Si se fija en 1 (una) la cantidad de iteraciones es interesante considerar algunos casos extremos para el tiempo de búsqueda: si se le asigna 0 (cero) tiempo, el comportamiento se reduce al de una versión aleatoria del algoritmo goloso; si se le asigna tiempo ilimitado, el comportamiento se reduce al de la estrategia de fuerza bruta. Se esperan soluciones cercanas a las óptimas en la mayoría de los casos, a un costo computacional menor que la estrategia exhaustiva y sin caer en situaciones patológicas como en la estrategia golosa; los parámetros de configuración y el azar son los nuevos medios a disposición.

La complejidad de peor caso es $O(itera * \text{mín}(W_c * t, 2^m))$ con $itera$ la cantidad de iteraciones, W_c la cantidad de combinaciones que la computadora «c» puede procesar por unidad de tiempo y «t» el tiempo durante el cual se le permitirá ejecutar a la etapa de búsqueda local; en la función *mín* queda plasmado que si el tiempo es suficientemente grande, predominará el análisis exhaustivo mientras que, si no, la búsqueda local explorará sólo una porción proporcional a t del espacio de búsqueda de la instancia.

2.6. Mejoras

Dejando de lado las estrategias y la complejidad inherente del problema, todavía quedan herramientas de las cuales se puede conseguir ayuda.

A continuación se presentan algunas ideas pensadas para facilitar el trabajo de optimización. Cada una tomará el grafo de sustituciones que representa a la especificación inicial, intentará detectar en él porciones que acepten un tratamiento eficiente, sin comprometer la calidad de las soluciones, y resolverá localmente esas situaciones favorables.

En cada caso se trata de procesos inmediatamente anteriores al análisis que efectúan las estrategias, complementarios con ellas y entre sí, para los que se describe su propuesta, supuestos, algoritmos, validez y complejidad temporal.

2.6.1. Descomposición en componentes conexas

Se pretende dividir un problema grande en subproblemas más pequeños, para que la resolución independiente de cada una de las partes solucione el problema original: dividir y conquistar. La descomposición de un grafo en componentes conexas es un preproceso conveniente en los problemas combinatorios

que lo admiten, dado que en condiciones favorables es capaz de evitar esfuerzo innecesario en un análisis exhaustivo. Para explicar esta mejora convienen más definiciones.

Definición 32 (componente conexa)**:** un grafo no dirigido se dice *conexo* si cada par de nodos se puede unir por un camino y *disconexo* si no. Se le dice *componente conexa* o *componente* de un grafo simple a un subgrafo conexo maximal.

Retomando la descripción, se recibe H el g.s. original y tras procesarlo se consigue $H_1, \ldots, H_k$ una descomposición con cada subgrafo más simple si $k > 1$. Ya que la descomposición del grafo y la combinación de resultados se realizan en tiempo polinomial, en el peor caso la demora adicional es insignificante, mientras que en el mejor se pueden conseguir ganancias sumamente importantes.

Más en detalle, se toma el original $H =< V, X >$ y se detectarán las componentes $G_1 =< V_1, X_1 >, \ldots, G_k =< V_k, X_k >$ de su grafo no dirigido subyacente, generando una familia $\{V_1, \ldots, V_k\}$ que parte V. Con esta partición, se definen nuevos hipergrafos $H_1 =< V_1, X'_1 >, \ldots, H_k =< V_k, X'_k >$ tales que si $e = (o_e, D_e) \in X$ y $\{o_e\} \cup D_e \subseteq V_j$ para algún $1 \leq j \leq k$ entonces $e \in X'_j$ y además los X'_j son los mínimos conjuntos con esos elementos. La familia de hiperarcos $\{X'_1, \ldots, X'_k\}$ parte X.

Teorema 6: descomponer el g.s. en componentes conexas permite analizar exhaustivamente todo el grafo sin más esfuerzo, y probablemente con menos.

Demostración: dado H el g.s. original y su descomposición en $H_1, \ldots, H_k$ no nulos, pueden darse dos casos: $k = 1$ ó $k > 1$. Si $k = 1$ entonces $H_1 = H$ y analizarlo demandará el mismo esfuerzo que sin haber descompuesto el grafo. Si $k > 1$ entonces $H_1, \ldots, H_k$ son todos más pequeños que H y, puesto que la estrategia de fuerza bruta demanda 2^{m-1} esfuerzo para una instancia de tamaño m, es posible ver que la descomposición insumirá esfuerzo

$$\sum_{i=1}^{k} 2^{m_{H_i}-1} \leq \max_{i \in \{1..k\}} 2^{m_{H_i}} \leq 2^{m_H - 1}$$

dado que $\sum_{i=0}^{j} 2^i < 2^{j+1}$, $\sum_{i=1}^{k} m_{H_i} = m_H$ y es aditivo el esfuerzo de analizar cada componente por separado. □

En la implementación esta mejora aparece bajo el nombre `splitInCC(...)`; explicación y pseudocódigo del algoritmo, justificación de su validez y análisis de complejidad temporal se encuentran en [NSS94] con el nombre de «Improved Algorithm 1» («Algoritmo mejorado 1»). Se trata de una variante acelerada de la detección de componentes fuertemente conexas de [HT73]. Emplear el algoritmo en grafos no dirigidos devuelve la descomposición deseada en componentes conexas.

2.6.2. Descomposición en componentes fuertemente conexas

La partición en componentes conexas sirve como un primer paso que se queda corto; con el mismo esfuerzo es posible conseguir una descomposición en clases iguales o más pequeñas que las anteriores. Para explicar la nueva descomposición, más definiciones.

Definición 33 (componentes fuertes y débiles)**:** un digrafo se dice *fuertemente conexo* si cada par de nodos se puede unir por un camino dirigido y *débilmente conexo* si cada par de nodos se puede unir por un camino simple en el grafo no dirigido subyacente. Se le dice *componente fuertemente conexa* (*scc*, por sus siglas en inglés) de un digrafo a un subgrafo fuertemente conexo maximal; análogamente se definen las *componentes débilmente conexas* (*wcc*).

De las definiciones se desprende que la descomposición en componentes conexas de la mejora anterior coincide con una descomposición en wcc, que el grafo de sólo un nodo es fuertemente conexo, que un grafo fuertemente conexo es también débil, y que una scc es un subgrafo débilmente conexo del grafo subyacente, aunque no necesariamente maximal. Como consecuencia, la anterior descomposición puede encontrar en cada componente más que una scc pero no menos; dicho de otro modo, una descomposición en scc arroja siempre componentes más pequeñas o iguales. Por ejemplo, en una wcc podrían encontrase dos scc unidas por un arco.

En el tránsito hacia esta nueva descomposición más fina hay que ser cuidadoso. La partición en wcc lograba dividir el grafo original en subgrafos disjuntos, prácticos para trabajar y explicar; ahora se podrán encontrar hiperarcos que vinculen distintas scc y habrá que tratarlos con especial cuidado. Partiendo el conjunto de nodos según las scc del digrafo subyacente, y dado un arco del grafo original, hay que considerar los casos:

1. si origen y destino están todos en una misma scc, el arco deberá ser considerado cuando se analice la componente en que figure el nodo origen;

2. si algunos elementos del destino están en la misma scc que el origen y otros no, no se tiene garantía de que el arco pueda ser ignorado en el análisis de la componente del nodo de origen y, por lo tanto, el arco también deberá ser considerado como si el destino completo estuviera en la misma componente;

3. si todos los nodos del destino están en scc distintas de la del nodo origen, el arco podrá ser ignorado en el análisis de la componente del nodo origen y será analizado por separado.

Volviendo al ejemplo de recién, dos scc unidas por un arco ahora serán analizados en tres momentos distintos: cada scc por su lado y el arco que las une por separado. Esto es, resultarán dos tipos de arcos: los que forman parte de componentes fuertes y los que no, los primeros se analizan localmente por cada componente y los segundos se analizan por separado, más adelante se verá cómo. Cualquiera sea el tipo de arco, siempre será analizado de una u otra forma.

En general, dado $H =< V, X >$ el g.s. original, sobre la partición $V_1, \ldots, V_k$ de V según las scc del grafo subyacente, se construye una nueva partición $X_1, \ldots, X_k, X_{k+1}$ de los hiperarcos en X tal que, si $1 \leq i \leq k$, en X_i se encuentran los arcos de los casos 1 y 2, si su origen está en la scc que tiene por nodos a V_i, y en X_{k+1} los del caso 3, que son todos los demás. Ello conduce a la descomposición final del g.s. H en $H_1 =< V'_1, X_1 >, \ldots, H_k =< V'_k, X_k >$, $H_{k+1} =< V'_{k+1}, X_{k+1} >$, con $V_i \subseteq V'_i$ y $V'_i - V_i$ los nodos que agrega el caso 2 para $1 \leq i \leq k$, y V'_{k+1} los nodos involucrados en el caso 3 de algún arco.

Teorema 7: descomponer el g.s. en scc permite analizar exhaustivamente todo el grafo sin más esfuerzo que la mejora anterior, y probablemente con menos.

Demostración: como por definición las wcc engloban a las scc, alcanza con observar las posibilidades de una sola componente general. Dada una wcc y su descomposición en $H_1, \ldots, H_k$ no nulos y H_{k+1}, pueden darse dos casos: por un lado, que $k = 1$ y H_{k+1} sea el grafo nulo; por el otro, que $k > 1$ ó H_{k+1} no sea el grafo nulo. Si $k = 1$ y H_{k+1} es el grafo nulo entonces $H_1 = H$ y analizarlo demandará el mismo esfuerzo que sin haber descompuesto el grafo. Para las restantes posibilidades hay que tener presente que la estrategia de fuerza bruta demanda 2^{m-1} esfuerzo para una instancia de tamaño m. Si $k > 1$ ó H_{k+1} no es el grafo nulo entonces analizar exhaustivamente la descomposición demanda

$$\sum_{i=1}^{k} 2^{m_{H_i}-1} + esfuerzo(H_{k+1}) \leq 2^{m_H-1}$$

ya que el esfuerzo de procesar H_{k+1} se verá más adelante que es suficientemente pequeño, $\text{máx}_{i \in \{1..k\}}\, m_{H_i} < m_H$ y es aditivo analizar cada componente por separado; el razonamiento es análogo al de la descomposición en wcc. □

Igual que antes, en la implementación esta mejora aparece bajo el nombre `splitInCC(...)`; me remito a los comentarios previos para más detalles. La única diferencia con la mejora anterior es que ahora se emplea el algoritmo sobre un grafo dirigido.

2.6.3. Compresión de dahgs

El análisis de regiones acíclicas quedó vacante en las otras mejoras; ahora se busca darles un procesamiento eficiente (léase «polinomial») que además aprovecha la descomposición en scc.

Tras descomponer el g.s. se obtuvo un subgrafo, no necesariamente conexo, con todos los hiperarcos cuyo destino no tuviera ningún nodo en la misma scc que el origen. Este subgrafo debe ser acíclico por construcción, ya que en caso contrario existiría un camino dirigido desde el destino de algún arco hasta su origen, contradiciendo el criterio que, en primer lugar, exigía distintas componentes para ambos al momento de seleccionar los arcos.

Teorema 8: en un g.s. acíclico toda solución maximal consigue igual ahorro.

Demostración: dado un g.s. $H =< V, X >$ acíclico, sin perdida de generalidad se puede suponer que su grafo no dirigido subyacente es conexo. Voy a argumentar, primero, que para cualquier solución las hojas siempre están presentes en el grafo resultante y, luego, que además son los únicos nodos presentes; se sigue que la complejidad final del grafo será la misma y, por lo tanto, el ahorro idéntico para cualquier solución.

Como por definición las hojas no son origen de ningún arco y de las explicaciones en las Secciones 2.4.1 y 2.4.3 se desprende que sólo los orígenes de un arco pueden aspirar a ser reemplazados y, por lo tanto, eliminados del g.s., cualquiera sea la solución se sabe que las hojas permanecerán siempre en el grafo resultante.

Para demostrar que las hojas son los únicos nodos presentes en el grafo resultante, al margen de la solución escogida, en el Apéndice A.4 figura una demostración por absurdo bastante simple pero cargada con abundantes definiciones y notación poco atractiva. En su lugar, se puede explicar informalmente el razonamiento del siguiente modo: si se supone un nodo que no es hoja, deberá haber un arco que lo tenga por origen y un candidato que lo acompañe; sin embargo, una solución no sería maximal mientras este candidato persista como posibilidad y, si desaparece, es porque también el nodo original debe haber sido eliminado del grafo o bien el grafo no era acíclico en primer lugar, lo que de cualquier manera resulta una contradicción. □

Ya que en un grafo acíclico todas las combinaciones maximales de candidatos consiguen el mismo ahorro, cualquier algoritmo que provea una solución de ese tipo ahora dará garantía suficiente de alcanzar un resultado globalmente óptimo. En la implementación el algoritmo particular que efectúa esta mejora aparece bajo el nombre `dahgCompression(...)` y su pseudocódigo se presenta en la Figura 7.

```
Collection<Candidate>
dahgCompression(Collection<Candidate> acyclic)
{
  selectedCandidates <- new Collection<Candidate>();

  while not acyclic.isEmpty() {
    nextCandidate <- acyclic.getAny();
    selectedCandidates.add(nextCandidate);
    acyclic <- updateCandidates(nextCandidate, acyclic);
  }

  return selectedCandidates;
}
```

Figura 7: Compresión de dahgs.

Para justificar que el algoritmo finaliza su ejecución para toda entrada, basta observar que mientras haya candidatos uno se seleccionará y eliminará en cada actualización. Con una colección finita, eso es argumento suficiente si se recuerda que la actualización se justificó en la Sección 2.4.3.

Que cumple con su propósito, conseguir una colección maximal y determinística de candidatos, surge de no permitir la detención mientras queden elementos agregables y de la justificación de la actualización en la Sección 2.4.3, que consigue soluciones viables y determinísticas de mediar entre selecciones consecutivas de candidatos.

Por último, resta justificar su complejidad temporal. Si se mantiene el criterio usado hasta ahora, el de contar la cantidad de combinaciones de resultados analizados, surge del análisis que una única combinación es considerada, con lo que el algoritmo tiene una complejidad temporal $O(1)$ para cualquier tamaño de la entrada, independientemente de los sucesivos esfuerzos de actualización que aunque impacten en el desempeño no son de utilidad para comparar algoritmos.

3. Resultados

Además de planteos, teoremas y análisis de complejidad es conveniente poner a prueba la teoría. Esta sección examina las principales limitaciones de las estrategias, observa su desempeño comparado, más el de sus variantes si las tiene, y analiza la capacidad de las mejoras para complementar las estrategias.

Habrá dos dimensiones que serán observadas en cada prueba en que tenga sentido: la cantidad de evaluaciones y el error relativo. La cantidad de evaluaciones indica cuántas soluciones diferentes fueron analizadas antes de obtener un resultado, mientras que el error relativo precisa la calidad de la solución conseguida. No tiene sentido analizar el error de la estrategia exacta, ni la cantidad de evaluaciones del procedimiento goloso, que sólo analiza una única solución.

Por otro lado, dada la naturaleza combinatoria del problema conviene prestar atención al frecuente uso de escalas logarítmicas, tanto en los gráficos como en las configuraciones de las corridas.

Con relación a las estrategias, al estudiar los resultados de emplear GRASP se observará una poda de los datos, poda que descarta a las instancias analizadas exhaustivamente durante la fase exploratoria inicial de la estrategia. Ella responde a que las instancias suficientemente pequeñas no logran desplegar el comportamiento heurístico de la estrategia, arrojando resultados que se reducen a los que se hubieran conseguido mediante fuerza bruta, sin representar a la situación que se quiere analizar.

Las sucesivas pruebas fueron realizadas en una computadora equipada con un Mobile AMD Semptron 3000+ de 1,8Ghz y 512MB de RAM.

3.1. Límites de fuerza bruta

Idealmente se quisiera garantía de obtener siempre soluciones que sean óptimos globales. Sin embargo la única garantía conocida de soluciones globalmente óptimas se consigue mediante la estrategia de fuerza bruta, lo que impone una limitación importante: como el problema es de naturaleza combinatoria es esperable que exista un tamaño de las instancias a partir del cual deje de ser práctico esperar soluciones exactas.

Para indagar bajo qué condiciones la implementación de fuerza bruta deja de ser útil construyo dos familias de grafos, una en representación de los casos ralos compuesta por los grafos *ciclo*, otra en representación de los casos densos, compuesta por los grafos *completo*.

3.1.1. Grafo *ciclo*

Los grafos ciclo están constituidos por un único ciclo dirigido simple y se escribe C_n para indicar que el grafo tiene n nodos; son los hipergrafos con menos arcos que forman un ciclo simple de longitud $n + 1$ y cuyos destinos tienen la menor cardinalidad. En estos grafos el tamaño de las instancias coincide con n.

El objetivo de analizar el comportamiento de la estrategia con estos grafos es doble: observar si las predicciones temporales teóricas se cumplen y, si lo hacen, determinar (para una computadora dada) hasta qué tamaño de la entrada es viable analizar exhaustivamente.

Corro el algoritmo sobre sucesivos miembros de la familia, empezando por los más pequeños y creciendo hasta que el esfuerzo requerido para analizar com-

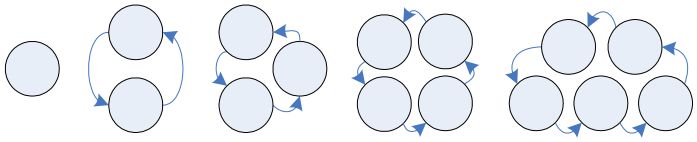


Figura 8: C_1 – C_5

pletamente una instancia exceda el tiempo que razonablemente se está dispuesto a esperar una respuesta. Después de casi 10 horas de cómputo el resultado de las corridas se ve en la Figura 9.

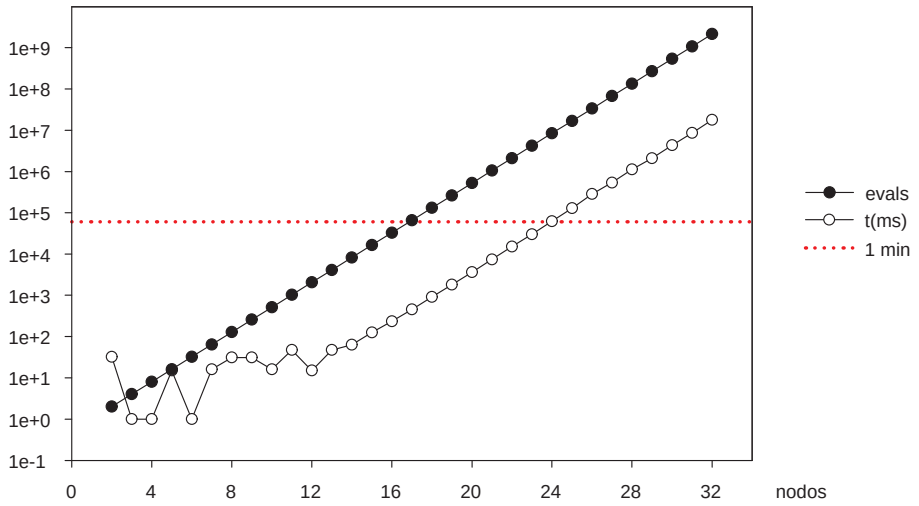


Figura 9: Ciclos – C_n

Se destaca:

- se confirma la predicción teórica;

  La teoría anticipaba un peor caso con complejidad temporal $O(2^m)$ para instancias de tamaño m y la prueba muestra que los grafos ciclo caen en el límite superior de esa clase de complejidad, lo que permite pensar que no son necesarias construcciones intrincadas para encontrar al peor caso. Al ratificar en la práctica que el esfuerzo se duplica con cada incremento del tamaño de estos grafos, se afirma la preocupación de que sólo instancias pequeñas podrán ser analizadas exhaustivamente.

- el tiempo insumido por las instancias pequeñas es volátil;

  Hasta alcanzar los 12-14 nodos los ciclos pequeños muestran insumir tiempos dispares, no correlacionados con la cantidad de evaluaciones. Hay que considerar que estas mediciones no superan la décima de segundo (1e+2ms) y que seguramente se ven influidas por problemas de ruido en la medición, producto de influencias externas (memoria caché, operaciones de e/s, precisión del cronómetro...), para llegar a la conclusión de que su manifestación puede ignorarse.

- se correlaciona el tiempo con las evaluaciones;

  Si se podan los resultados descartando tiempos por debajo de la décima de segundo se confirma que, para estos grafos, la duración del cómputo depende estrechamente de la cantidad de evaluaciones; observación atractiva porque si se la extrapola con optimismo para otros tipos de grafos permite suponer que, en instancias topológicamente semejantes, se consiguen buenas estimaciones a partir de los miembros pequeños y, con ello, se logra la capacidad de predecir el tiempo de cómputo. Ya que las estimaciones temporales son dependientes de la computadora, en general será preferible concentrar la atención sobre la cantidad de evaluaciones; sin embargo, un mínimo de detenimiento en el vínculo entre las evaluaciones y el tiempo que insumen ayuda a contextualizar las pruebas. Sobre todo, para acotar cuánto esfuerzo es razonable dedicar a una solución. En la computadora en que se realizan las pruebas se logran evaluar aproximadamente unas 125.000 soluciones por segundo, para este tipo de grafos.

- sólo instancias pequeñas pueden analizarse.

  Si el esfuerzo se duplica con cada aumento de tamaño y el tiempo es proporcional al esfuerzo, resta decidir cuánto tiempo se está dispuesto a esperar una respuesta para estimar el máximo tamaño admisible para una instancia, si es que se la quiere analizar con esta estrategia. Por ejemplo, suponiendo que una demora de un segundo es aceptable, en la computadora de referencia se puede suponer que instancias de esta familia con un tamaño no mucho mayor a 20 serán tratables; si se acepta un minuto, no mucho más de 25 y, con una hora, no más de 30.

3.1.2. Grafo *completo*

Los grafos completos están constituidos por todos los arcos posibles para el conjunto de nodos dado y los completos h-uniformes, en el caso de los hipergrafos hacia adelante, por todos los arcos posibles cuyo destino tenga cardinal exactamente h para el conjunto de nodos dado. Se escribe K_n para indicar al grafo completo de n nodos y K_n^h para indicar al grafo completo h-uniforme de n nodos. Para K_n^h el tamaño de las instancias es $n \cdot \binom{n-1}{h}$.

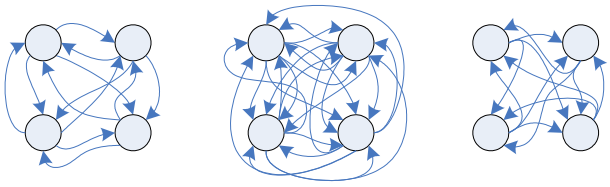


Figura 10: K_4^1 – K_4^3

El objetivo de analizar el comportamiento de la estrategia con estos grafos reside en equilibrar el conocimiento que se logró con los ciclos; mientras aquéllos incluían pocos arcos para una cantidad dada de nodos, éstos contienen la máxima posible. Se aspira a relativizar los resultados anteriores.

Al igual que antes, corro el algoritmo sobre sucesivas instancias hasta que el esfuerzo requerido exceda el tiempo que razonablemente se está dispuesto a

esperar una respuesta. Después de que algunas instancias demoraran, cada una, más de dos días de cómputo ininterrumpido, el resultado se ve en la Figura 11; la curva de referencia se explica más adelante.

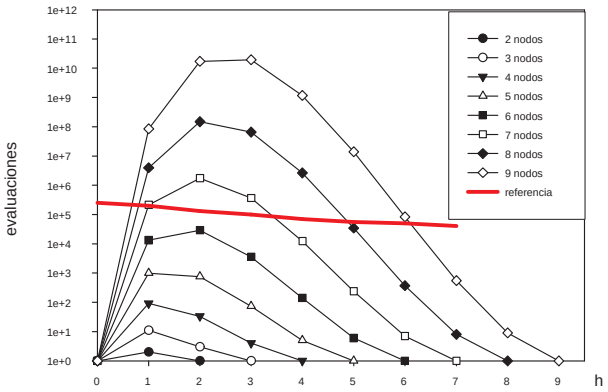


Figura 11: Completos – K_n^h

Se destaca:

- muy pocos nodos vuelven inabordable un grafo completo;

  Sólo hasta la serie de 8 nodos pudo procesarse con facilidad; la explosión de tamaño de estos grafos impide procesar subfamilias más allá de las analizadas. La excepción remarcable son las instancias K_n^{n-1}, tratables con complejidad lineal, que sin embargo no alteran en general el panorama. Aunque el tamaño de las instancias se define según la cantidad de hiperarcos, y no la de nodos, es útil recordar que apenas muy pocos nodos son suficientes para poder alcanzar un tamaño más allá de lo tratable.

- es necesario analizar otros aspectos.

  La figura es una primera aproximación que, aún careciendo de otra información importante, muestra de forma sencilla cómo evoluciona el esfuerzo insumido por cada instancia; su utilidad se limita a insinuar la naturaleza combinatoria de estos grafos. Para contrastar éstas con otras pruebas conviene adoptar una mirada centrada en el tamaño de las instancias.

Una pregunta que surge de querer comparar los grafos completos y los ciclos es si analizar distintas topologías presenta diferencias, pregunta que permite retomar la comparación con el tiempo de cómputo empleada antes y que llevará a explicar la curva de referencia de la Figura 11. Para comenzar a contestarla se observa la demora que acompaña a cada análisis exhaustivo, por más que el dato varíe entre computadoras; si se alcanza el objetivo de relativizar los resultados de los ciclos, el tiempo dejará de ser considerado una variable relevante en las pruebas que sigan.

Nuevamente, las instancias que demoraron menos de una décima de segundo tuvieron un comportamiento poco predecible y quedaron excluidas de los

comentarios; las instancias restantes exhibieron una correlación fuerte entre el tiempo y los parámetros del grafo, que se refleja en el Cuadro 1.

h	evals./seg.
1	200.000
2	130.000
3	100.000
4	70.000
5	55.000
6	50.000

Cuadro 1: Velocidades aproximadas

Se destaca:

- la velocidad no es constante;

  Si bien los datos son pocos, parecen indicar que las evaluaciones por unidad de tiempo dependen de h en este tipo de grafos.

- se podrían estimar cotas al tiempo que insumirá un análisis.

  Para la computadora en que se realizan las pruebas, es probable que la cota de 200.000 evals./seg. sirva para una estimación optimista y puede elegirse otro valor para el caso pesimista, dependiendo de la confianza que se requiera y de conocer mejor la distribución de valores.

La curva de referencia en la Figura 11 presenta, entonces, cuántas evaluaciones se efectuaron en el lapso de un segundo, para cada valor de h, y refleja los mismos valores que la Cuadro 1. Cualitativamente, indica que es esperable una cantidad del orden de las 100.000 ($1e{+}5$) evaluaciones en las mismas condiciones en que los ciclos arrojaban homogéneamente unas 125.000. En otras palabras, el cambio de grafos no modificó sustancialmente la velocidad de análisis.

Sin embargo, hay otras diferencias posibles. Una pregunta similar a la anterior es si el umbral de lo tratable también permanece invariante, inquietud análoga a averiguar si los grafos están igualmente bien representados por el peor caso. Para contestarla y, a la vez, observar los mismos resultados de la Figura 11 pero con más énfasis en el tamaño de las instancias, defino el cociente $c_i \in [0; 1]$, que indica cuán representativo resulta el peor caso. Si m_i es el tamaño y $evals_i$ la cantidad exacta de evaluaciones insumidas al analizar una instancia «i», la cota máxima estará dada por 2^{m_i-1} evaluaciones y

$$c_i = \frac{evals_i}{2^{m_i-1}}.$$

La evolución del cociente en K_n^h se ve en la Figura 12.

Se destaca:

- hay nuevas relaciones entre tamaño y complejidad;

  A diferencia de lo que sucedía en los grafos ciclo, en los completos se dan distintas relaciones entre tamaño y complejidad. En vez de permanecer constantemente igual a 1 (uno), a medida que aumenta la cantidad de nodos, e indirectamente el tamaño, el cociente tiende rápidamente a cero, alejándose del peor caso.

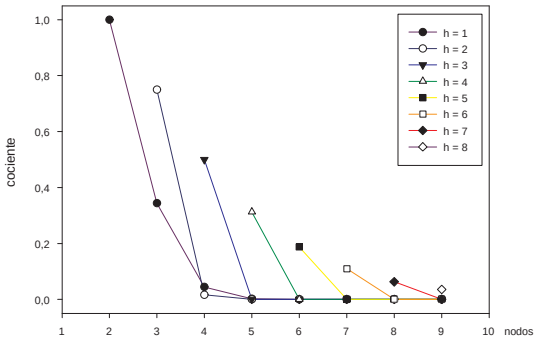


Figura 12: Evolución del cociente

- la intratabilidad práctica se mantiene.

  A pesar del alejamiento del peor caso en la evolución del cociente, las instancias de mayor tamaño se mantienen inabordables; el alejamiento es inseparable de un incremento combinatorio en el tamaño que neutraliza su efecto.

A igual cantidad de evaluaciones por unidad de tiempo, como es el caso de los ciclos y los completos, podrá haber una instancia de cada familia que resulten una tratable y la otra no, dado que las relaciones de cada tipo de grafo con el peor caso puede variar significativamente. Llegado este punto se sabe que el peor caso es poco representativo de algunos grafos y, gracias a ello, inconveniente como estimación única del esfuerzo; aporta una cota poco ajustada, razón por la cual el análisis de cada familia enfrentará distintos umbrales para el tamaño máximo de instancia tratable.

3.2. Límites de más pesado primero

Idealmente se quisiera obtener siempre soluciones instantáneas. La estrategia de más pesado primero está pensada específicamente para cumplir con este requisito, dejando de lado la pretensión de garantizar soluciones globalmente óptimas; se consiguen soluciones rápidas aunque puedan ser imperfectas.

Para indagar sobre la calidad de las soluciones que ofrece la implementación golosa construyo una familia de grafos que representa a los casos que explotan la debilidad del método.

3.2.1. Grafo *estrellado*

Los grafos estrellados se inspiran es los grafos *estrella* y están constituidos por un nodo central con un único arco que tiene por destino a todos los demás, y otros varios con un único arco que sólo tiene por destino al primero. Escribo E_n para indicar que el grafo tiene n nodos. En estos grafos el tamaño de las instancias coincide con n.

El objetivo de analizar el comportamiento de la estrategia con estos grafos es observar la calidad de las soluciones que se consiguen. Se pretende conse-

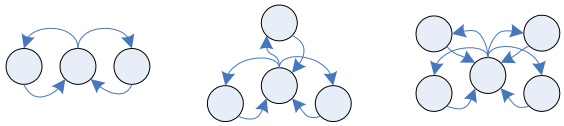


Figura 13: E_3 – E_5

guir malas soluciones y con ello abandonar la esperanza de usar esta estrategia indiscriminadamente.

Corro el algoritmo sobre sucesivos miembros de la familia y comparo los resultados con los que se obtendrían por medio de un algoritmo exacto. Para lograr dirigir el comportamiento de la estrategia, al nodo central le asigno una ganancia mayor a la del resto. La evolución del error relativo de los resultados se ve en la Figura 14.

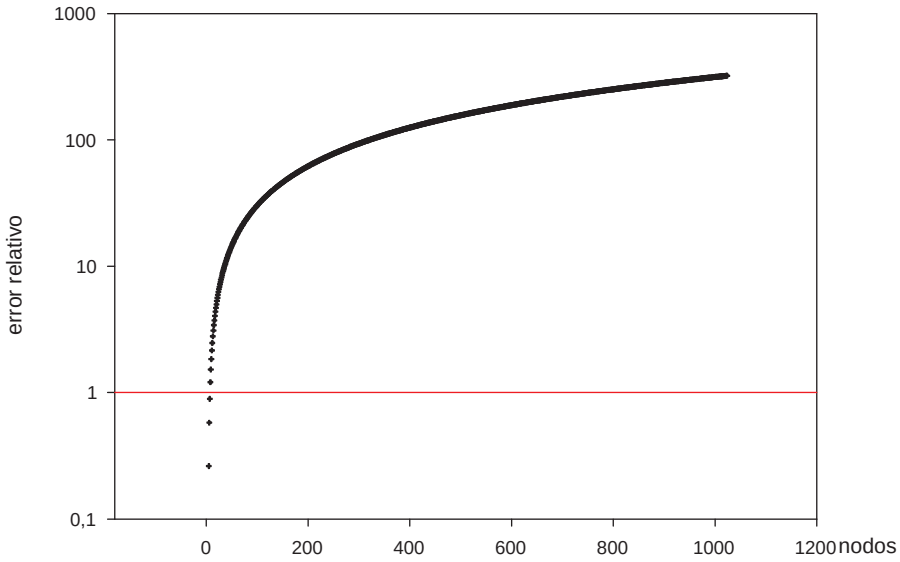


Figura 14: Evolución del error relativo

Se destaca:

- el error relativo es muy alto;

  Por encima de 1 (uno) el error relativo ya es muy alto y la calidad de las soluciones es de dudosa utilidad. Sería fácil demostrar que, además, puede volverse tan grande como uno quiera.

- pudieron procesarse instancias mucho mayores.

  A diferencia de las limitaciones que encuentra la estrategia de fuerza bruta, instancias con tamaño en los cientos de arcos pudieron procesarse sin complicaciones muy velozmente.

3.3. Límites de GRASP

De esta estrategia se quisiera obtener siempre soluciones con calidad cercana al óptimo pero insumiendo un pequeño esfuerzo. Sin embargo, como el criterio empleado por la heurística para restringir los candidatos seleccionados se guía por el mismo principio que la estrategia golosa, es esperable que se vea influenciado negativamente ante iguales situaciones desfavorables.

Para someter la estrategia a condiciones adversas y estudiar la degradación de su desempeño empleo las mismas instancias usadas para explorar los límites de más pesado primero.

El objetivo es observar la calidad de las soluciones que se consiguen y comprobar cómo responde a instancias grandes. Se pretende conseguir un desmejoramiento progresivo de la calidad y detectar problemas vinculados al tamaño de las instancias; es una forma de conocer qué exigencia es capaz de tolerar la estrategia.

En las pruebas se configuró la heurística con 1 (un) segundo de umbral y 128 iteraciones[5], empleando las mismas instancias que en más pesado primero. Después de más de 1 (un) día de cómputo, el resultado de las corridas se ve en la Figura 15.

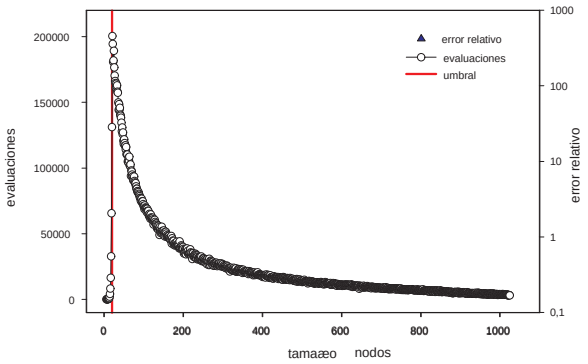


Figura 15: Esfuerzo y error relativo

Se destaca:

- no se encontraron diferencias de calidad con el caso exhaustivo, las soluciones fueron todas exactas;
- el umbral de tiempo condiciona el esfuerzo;

  El umbral detiene el crecimiento exponencial del esfuerzo y, conforme otros factores como la e/s influyen en el desempeño de la estrategia, induce a la paulatina disminución de la cantidad de evaluaciones.

[5]Parámetros elegidos arbitraria pero consistentemente.

- ninguna instancia sobrepasó la capacidad de cómputo;

  Los factores no considerados que influyen en el tiempo de procesamiento se compensan con la disminución progresiva del esfuerzo para no exigir en exceso a la computadora.

- pudieron procesarse sin inconvenientes instancias de gran tamaño, al igual que con la estrategia más pesado primero.

3.4. Mejoras

Sabiendo que la tarea por delante de las estrategias puede resultar muy ardua, las mejoras aspiran a simplificar algunas porciones del trabajo. Idealmente se quisiera que su ámbito de aplicación alcanzara a todo el grafo, para cualquier tipo de grafo, reduciéndolo a una instancia mucho más pequeña; sin embargo es esperable que existan subgrafos que no puedan ser reducidos por las mejoras.

Para indagar bajo qué condiciones y con cuánta efectividad las mejoras impactan en el desempeño de las estrategias construyo una familia de grafos que representa al hipergrafo promedio.

3.4.1. Grafo *aleatorio*

Los grafos aleatorios son grafos generados por un proceso aleatorio; se comienza con un conjunto de n nodos al que se agregan arcos al azar. En este trabajo se incluye cada hiperarco con probabilidad p, independientemente y sin reposición, extendiendo para los g.s. el modelo $\mathcal{G}(n, p)$ descrito en [Bol98, Die00]. Un atractivo del este modelo es que, por medio de análisis probabilísticos [AS92], se conocen para los grafos que produce interesantes propiedades asintóticamente válidas que ayudan a guiar y enmarcar teóricamente las pruebas realizadas.

El objetivo de analizar múltiples corridas sobre estos grafos es observar el impacto cualitativo y cuantitativo de las mejoras en el desempeño esperado de una estrategia.

Corro cada estrategia con y sin mejoras sobre sucesivos miembros de la familia, para detectar diferencias. Para n una cantidad dada de nodos, construir hipergrafos $R \in \mathcal{G}(n, p)$ demanda una decisión respecto de los valores de p a emplear; es esperable que distintos valores arrojen instancias cuyas propiedades las conviertan en más o menos idóneas para las pruebas a realizar. Como la cantidad de posibles arcos en un g.s. es $c = n \cdot (2^{n-1} - 1)$ un número exponencial, a menos que p tienda a cero tan rápidamente como crece c cuando n tiende a infinito, la cantidad esperada de arcos será también exponencial. A su vez, los grafos densos, digamos con tamaño en el orden o por encima de n^2, difícilmente sean susceptibles de ser influidos por las mejoras, con lo que cualquier valor de p conviene que arroje arcos por debajo de esa cota. En una primera aproximación defino $p = f \cdot s$, con f una función de crecimiento sobre la que todavía resta profundizar y $s = 2^{-n}$ un factor de escala que controla el crecimiento exponencial de los hipergrafos. Experimentalmente $f = k \cdot \ln(n)$ resultó satisfactoria, con k una constante, función adoptada siguiendo el enfoque de [FM95] sobre el trabajo de [ER60]. De tratarse de grafos simples y mientras $n \to \infty$, por encima de este umbral las instancias tienen probabilidad 1 (uno) de ser conexas y 0 (cero) por debajo. Adicionalmente, para contar con instancias por encima y por debajo del umbral, se introdujo una variable $d \in [0; 1]$ a la que se llamará *densidad* y

que permite controlar más finamente la proporción de arcos de las instancias. Su participación es tal que, si $d = \frac{1}{2}$, no se altera el umbral; a tal fin, se definió

$$p = k \cdot \ln^{2d}(n) \cdot 2^{-n}.$$

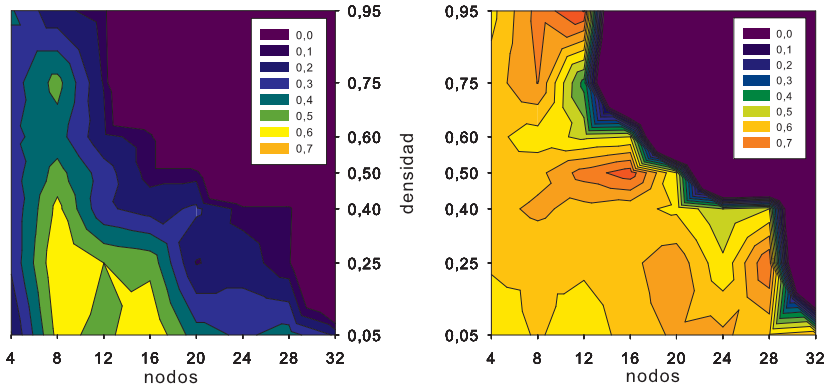
Para $d \in \{0{,}05; 0{,}25; 0{,}4; 0{,}5; 0{,}6; 0{,}75; 0{,}95\}$, $k = \frac{1}{2}$, n de 4 a 32 en incrementos de a 4 y 32 instancias de cada combinación, 1792 casos en total, los resultados se presentan por separado para cada estrategia. Con una demora para construir cada instancia exponencial en n, las aproximadamente 3 horas insumidas por cada uno de los 224 grafos con $n = 32$ (más de 25 días de cómputo) explica que no resulte práctico el enfoque empleado para valores más grandes de n.

3.4.2. Mejoras y fuerza bruta

Al evaluar el impacto de las mejoras con esta estrategia interesa conocer qué diferencia hubo en cuanto a la cantidad de evaluaciones necesarias para procesar las instancias.

Un primer resultado global es la disminución relativa de la cantidad de evaluaciones, que se consigue cocientando la disminución absoluta acumulada para todas las instancias sobre el total de evaluaciones. La diferencia fue una disminución relativa global de $\frac{108472260-108469593}{108472260} \approx 2,459e-05$.

Para ayudar a explicar el impacto que tienen las mejoras en casos individuales sobre el resultado global, se tiene la Figura 16.



(a) casos afectados (b) magnitud podada

Figura 16: Mejoras y fuerza bruta

En la parte (a) se grafica qué proporción de casos resultó afectada por alguna mejora, al margen de su magnitud, y permite observar cómo a medida que aumenta la cantidad de nodos, o la densidad de los grafos, disminuye la capacidad de las mejoras para simplificar estas instancias. En la parte (b) se grafica la magnitud relativa de la disminución de esfuerzo que se consiguió en promedio, restringiendo los casos a aquellos en los que sí se registró una diferencia, es decir, podando los valores nulos; se observa que la distribución de la magnitud es bastante homogénea.

Los indicadores estadísticos básicos para la disminución relativa de la cantidad de evaluaciones, sin podar y podados, se ven en el Cuadro 2; valores altos

son mejores, salvo para el desvío. Cerca del 80 % de casos sin mejora explica la diferencia entre el promedio y la mediana, en el caso sin poda, y justifica el interés por podar los valores nulos.

	sin poda	con poda
máximo	0,958	0,958
mínimo	0,000	0,250
promedio	0,120	0,564
mediana	0,000	0,500
desvío	0,241	0,155

Cuadro 2: Indicadores básicos

Notar la concordancia entre la homogeneidad de valores en la parte (b) de la Figura 16 y los indicadores podados, a diferencia de lo que sucede sin podar.

Combinando la lectura de las figuras y los indicadores se explica cómo es posible que el empleo de las mejoras aporte una disminución tan pequeña del esfuerzo global, tan lejos de la proporción de casos con mejoras y de la magnitud podada promedio: sólo los casos más pequeños y menos influyentes se vieron afectados. En otro planteo relacionado, si se fija en 100.000 evaluaciones una cantidad arbitraria que se supone representativa del análisis que puede efectuarse en un lapso humanamente perceptible, no hubo ninguna instancia donde la disminución absoluta de esfuerzo alcanzara ese valor.

3.4.3. Mejoras y más pesado primero

Al evaluar el impacto de las mejoras con esta estrategia interesa conocer qué diferencia hubo en la calidad de las soluciones.

La comparación de las corridas con y sin mejoras arrojó como resultado soluciones idénticas; no se encontraron diferencias.

3.4.4. Mejoras y GRASP

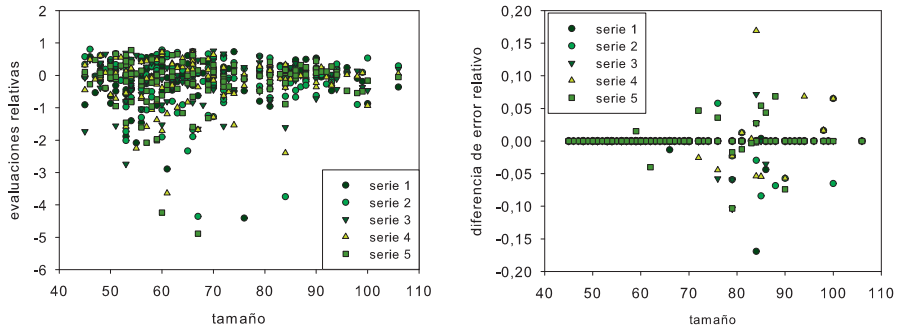
Al evaluar el impacto de las mejoras con esta estrategia interesa conocer qué diferencias hubo en cuanto a la cantidad de evaluaciones necesarias para procesar las instancias, y a la calidad de las soluciones.

A diferencia de otros casos, la variabilidad que introduce el azar en estas pruebas produjo resultados significativamente distintos tras repetir las ejecuciones. Con el fin de resaltar esas diferencias se corrieron varias series de ejecuciones y se decidió adaptar las variables analizadas para facilitar su presentación: se optó por combinar en una sola magnitud, el tamaño, lo que en otras pruebas figuraba desglosado según densidad y cantidad de nodos.

La comparación de 5 series de corridas con y sin mejoras, para GRASP configurado con 1 (un) segundo de umbral y 128 iteraciones, insumió un total de algo más de 8 horas de cómputo y se ve reflejada en la Figura 17.

En la parte (a) se grafica la diferencia relativa en cantidad de evaluaciones, descartando [6] las instancias que se lograron analizar exhaustivamente dentro del umbral de tiempo. Valores positivos son preferibles pues significan que al aplicar las mejoras propuestas se obtuvo una disminución del esfuerzo. No se advierte

[6] Ver explicación al comienzo de esta sección.



(a) esfuerzo relativo

(b) error relativo

Figura 17: Mejoras y GRASP

una clara tendencia a aumentar ni disminuir el esfuerzo, aunque sí se destaca la existencia de casos en los que el incremento de esfuerzo (valores negativos) es varias veces mayor, en magnitud, que la máxima disminución (valores positivos).

En la parte (b) se grafica la diferencia del error relativo, con igual poda que en la parte (a). Valores positivos son preferibles pues significan que al aplicar las mejoras se obtuvo un aumento en la calidad de la solución. Se destaca que la mayoría (más del 90 %) de los casos no arrojó diferencia y que no se advierte tendencia a aumentar ni disminuir el error.

Un complemento a la Figura 17 se consigue observando los indicadores estadísticos básicos del Cuadro 3. El «esfuerzo» se corresponde con la parte (a) de la figura y el «error» con la (b).

indicador	esfuerzo	error
máximo	+0,900	+0,060
mínimo	−4,300	−0,170
promedio	−0,524	−0,052
mediana	−0,071	−0,023
desvío	+1,743	+0,051

Cuadro 3: Mejoras y GRASP

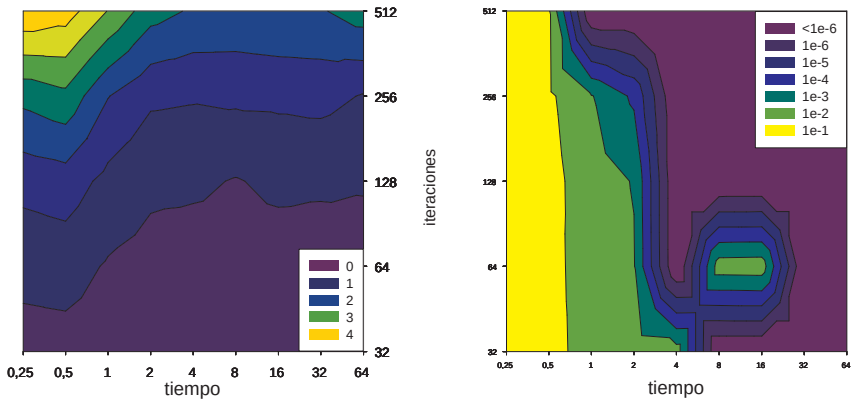
3.5. Comparación de variantes

Con una multitud de variantes para elegir, antes de comparar estrategias entre sí se examina el comportamiento a su interior. De las estrategias contempladas, GRASP es la única que ofrece esta posibilidad. A continuación, todas las pruebas se realizan aplicando las mejoras propuestas.

3.5.1. Variantes GRASP

Como se introdujo en la Sección 2.5.3 y detalla en el Apéndice C esta estrategia dispone de dos parámetros que caracterizan las variantes admisibles: la cantidad de *iteraciones* durante las cuales se repite el proceso y el *tiempo* estimado durante el cual se permite ejecutar a cada iteración.

Las siguientes pruebas están pensadas para reconocer el esfuerzo demandado por las soluciones y observar la calidad de los resultados, según cada combinación de los parámetros. En representación del hipergrafo promedio, en todos los casos se corrió la estrategia sobre el lote completo de instancias aleatorias también usado en la Sección 3.4 durante el análisis de impacto de las mejoras. El total de los datos del lote, para una combinación de los parámetros, se condensa a fin de resaltar en desempeño de la variante. Tras más de 12 días de cómputo, los resultados de las corridas se observan en la Figura 18.



(a) Esfuerzo relativo

(b) Error relativo

Figura 18: Variantes GRASP

En la parte (a) se grafica cuánto esfuerzo demandó a la variante analizar el lote, medido en veces que se realizó el mismo esfuerzo (cantidad de evaluaciones) que el promedio de todas las variantes y una vez podados[7] los datos; aunque arbitraria, la referencia es útil para contrastar el esfuerzo relativo de cada variante. Como es de esperar, en el gráfico se ve que a medida que aumenta el número de iteraciones también aumenta el esfuerzo insumido y, también, que el ritmo con que se produce ese aumento acompaña el crecimiento exponencial de las iteraciones. Menos evidente resulta la comparativamente leve influencia que ejerce el aumento del umbral de tiempo en el aumento del esfuerzo.

En la parte (b) se grafica el error relativo, también con poda. Allí se insinúa la confirmación de que, al aumentar el número de iteraciones o el umbral de tiempo, los errores decrecen.

3.6. Comparación de estrategias

Dado que se dispone de distintas estrategias surge la inquietud de observar cómo se comparan entre sí. Se quisiera recolectar información empírica que permita estimar el desempeño en cada caso.

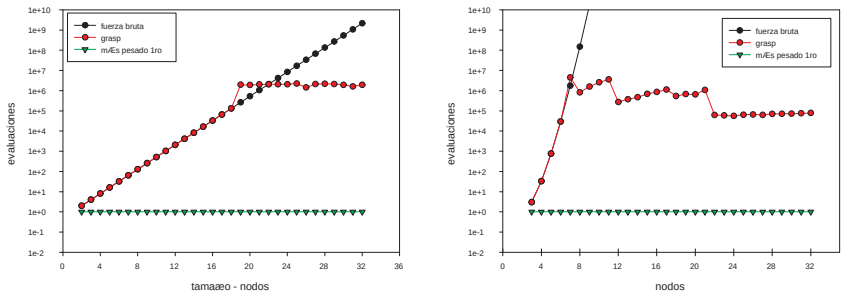
Reconociendo que distintas familias de grafos favorecerán o perjudicarán a cada estrategia, se realizaron todas las pruebas atractivas para cada tipo de grafos definido hasta ahora, ya que cada uno representa un caso interesante de por sí; los resultados y observaciones se presentan junto a cada familia.

[7] Ver explicación al comienzo de esta sección.

Todas las pruebas siguientes fueron ejecutadas aplicando las mejoras y, en cada oportunidad en que GRASP fue puesto a prueba, se consideró arbitrariamente como representante a la variante con 1 (un) segundo de umbral de tiempo y 128 iteraciones.

3.6.1. Grafos ciclo y completo

Los ciclos representan los casos ralos y los completos los densos; sus descripciones figuran en las Secciones 3.1.1 y 3.1.2 respectivamente. Mientras que las instancias empleadas en las pruebas con ciclos fueron las mismas que en los casos anteriores, en las pruebas con completos se empleó a la subfamilia K_n^2 añadiendo instancias, inabordables para fuerza bruta, para las cuales se obtuvieron sólo los valores que fue práctico conseguir en ejecuciones reales. Interesa estudiar el esfuerzo insumido por cada estrategia y no así la calidad de las soluciones, que dado el modo en que están construidas las instancias, en todos los casos arrojan resultados exactos. Los resultados de las corridas se ven en la Figura 19.



(a) Ciclos – C_n (b) Completos – K_n^2

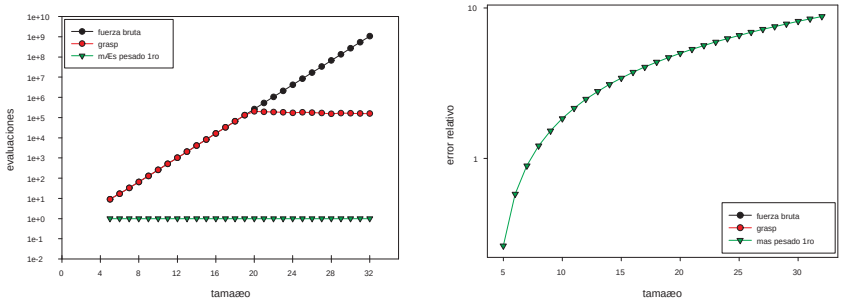
Figura 19: Comparación de estrategias (grafos C_n y K_n^2)

En la parte (a) se destaca el quiebre que produce el umbral de tiempo en GRASP; antes del umbral la estrategia se comporta igual que fuerza bruta y, después, las evaluaciones se estacionan en una meseta. Inmediatamente a continuación del quiebre la cantidad de evaluaciones se advierte mayor que en el análisis exhaustivo pero, a medida que aumenta el tamaño de las instancias, el menor esfuerzo se observa marcadamente del lado de la heurística. En la parte (b) se observa otra manifestación del mismo fenómeno que en la parte (a), con saltos antes de parecer alcanzar un punto de equilibrio.

3.6.2. Grafos estrellados

Representan los casos problemáticos para las estrategias no exhaustivas; su descripción figura en la Sección 3.2.1 y se emplean las mismas instancias que en pruebas anteriores, restringiendo el tamaño a los casos tratable por fuerza bruta. Interesa estudiar el esfuerzo insumido por cada estrategia y la calidad de las soluciones obtenidas. Los resultados se ven en la Figura 20.

En la parte (a) se observa el mismo fenómeno que en la última comparación. En la parte (b), fuerza bruta arroja siempre soluciones exactas mientras que las otras estrategias repiten los resultados analizados en pruebas anteriores.

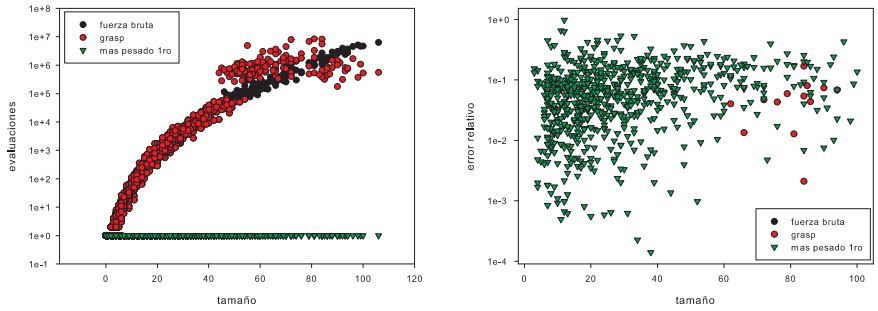


(a) Esfuerzo (b) Error relativo

Figura 20: Comparación de estrategias (grafos estrellados)

3.6.3. Grafos aleatorios

Representan a los casos promedio, su descripción figura en 3.4.1 y se emplean las mismas instancias que en otras pruebas. Interesa estudiar el esfuerzo insumido por cada estrategia y la calidad de las soluciones halladas. Los resultados de las corridas se ven en la Figura 21.



(a) Esfuerzo (b) Error relativo

Figura 21: Comparación de estrategias (grafos aleatorios)

En la parte (a) se destaca como, inmediatamente a partir del punto de quiebre determinado por el umbral de tiempo, la heurística insume en total más esfuerzo que el análisis exhaustivo, y no menos, dada la acumulación de evaluaciones de todas las iteraciones. Antes del quiebre la heurística se comporta igual que la estrategia exhaustiva, después parece alcanzar una meseta.

En la parte (b) se destaca lo abundante del error en la estrategia golosa frente a lo escaso que resulta en la heurística. La ausencia de errores en GRASP para las instancias más pequeñas se entiende observando la parte (a) de la figura.

4. Discusión

Esta sección recoge los aspectos salientes de los resultados y profundiza en los puntos que, sometidos a consideración, permiten lograr un entendimiento más acabado. Comienza repasando las pruebas realizadas y finaliza con algunos aspectos interesantes que no se desprenden directamente de ninguna prueba en particular.

4.1. Límites de las estrategias

De las pruebas se desprende que las limitaciones de fuerza bruta y más pesado primero las descalifiquen como estrategias de uso general. Si bien en la práctica pueden encontrarse instancias para las cuales arrojen resultados satisfactorios, sólo GRASP logró evitar las situaciones más desfavorables de las otras estrategias. Mejores implementaciones o computadoras más rápidas no hacen prever cambios.

Sin embargo, los resultados arrojados por las corridas de la heurística tampoco ofrecen una garantía absoluta. Por empezar, que las pruebas realizadas no consigan desacreditar a la estrategia no descarta que pueda existir otra familia de grafos que lo logre. Además, aún insistiendo en la familia empleada, la disminución paulatina que se observa en la cantidad de evaluaciones al aumentar el tamaño de los grafos permite sospechar que, para grafos suficientemente grandes, habrá un punto a partir del cual la calidad de las soluciones comenzará a verse comprometida. Incluso, es razonable suponer que para cualquier familia de grafos GRASP se verá comprometido cuando las instancias se agranden lo suficiente.

4.2. Mejoras

Se destaca como primera observación que las mejoras, en general, no reportaron ninguna ventaja significativa por sobre el desempeño de las estrategia, sea con relación al esfuerzo o a la calidad de las soluciones.

Entre las causas que explican este magro avance se encuentra la ausencia de instancias con una estructura elaborada, debida a la cual en ningún caso aparecieron, al menos, dos CFC de tamaño comparable al de la instancia que las contiene. En esta situación se justifica que los resultados no ahonden en el origen de cada pequeño cambio que, de hecho, sólo provienen de la compresión de dahgs.

Una causa apenas sospechada, y sin confirmación suficiente en los resultados, que podría agravar el escaso impacto de las mejoras radica en la posibilidad de que los grafos aleatorios, que son los únicos susceptibles de mostrar cambios, rijan su crecimiento por una función de probabilidad inapropiada. Si al aumentar la cantidad de nodos, su tamaño esperado aumentara demasiado velozmente se observaría, como sucede, que a medida que se agregan nodos las instancias son cada vez más refractarias de las mejoras.

En otro plano resulta sugerente que la calidad de las soluciones de más pesado primero haya resultado inmutable a las mejoras. Sería interesante contar con una demostración que justifique este comportamiento o, más en general, averiguar bajo qué condiciones se obtiene esta conducta.

Del mismo modo, otras explicaciones también están ausentes: ¿por qué motivo la cantidad de evaluaciones de GRASP parece aumentar al aplicar las mejoras? ¿Por qué en la heurística se observa poca dispersión de valores en la disminución de esfuerzo y no así el aumento?

Por último resta reconocer que, a pesar de todo, el pequeño esfuerzo computacional de aplicar las mejoras avala su uso de manera prácticamente irrestricta. Aunque sin grandes logros, y en particular considerando el esfuerzo insumido por la estrategia exhaustiva, el empleo de mejoras abre un camino del cual queda mucho por explorar; otras ideas podrían sumarse, así como otras familias de instancias podrían permitir apreciar más claramente el beneficio de utilizar las que hoy hay.

4.3. Comparación de variantes

El estudio del desempeño de las variaciones de GRASP anima la imaginación; surge la inquietud por contestar a «¿Qué combinación de parámetros dará las mejores soluciones?». Me mantendré alejado de la tentación de responder a esa pregunta. Como regla general, la calidad de los resultados dependerá de los recursos dedicados, quedando en manos del usuario decidir cuánto está dispuesto a esperar una solución.

También en sus manos está la posibilidad de asemejar la heurística a las otras estrategias, mediante combinaciones de valores extremos de los parámetros. Es decir, se puede entender que la heurística subsume a las demás alternativas.

4.4. Comparación de estrategias

Descartadas las estrategias exhaustiva y golosa para el uso general, sólo la heurística permanece como una alternativa viable; es momento de evaluar cualitativamente su desempeño, tanto desde el esfuerzo que le insume encontrar soluciones, como desde la calidad de sus resultados.

Al respecto, las pruebas confirman la capacidad de GRASP para limitar el esfuerzo insumido ya que, en cada ocasión en que la estrategia superó el umbral durante el cual tenía permitido ejecutar, a continuación se adaptó satisfactoriamente a la imposición de una cota de esfuerzo.

La limitación del esfuerzo, no obstante, no encontró en las pruebas situaciones en las que la calidad de las soluciones resultara seriamente comprometida sino, más bien, lo contrario. Diferencias con los valores óptimos fueron la excepción y, cuando se dieron, se mantuvieron siempre dentro de márgenes admisibles.

El único inconveniente destacable de la heurística fue que, para las instancias que requerían un análisis exhaustivo ligeramente por encima de lo que el umbral de ejecución les permitía, el esfuerzo insumido superó al empleado por fuerza bruta sin siquiera garantizar la óptima calidad de los resultados; situación que da origen a posibles cambios futuros en la implementación de la estrategia.

Por otra parte, poniendo en consideración no ya las estrategias sino las pruebas, es importante remarcar la consistencia que se observa en los resultados, con independencia de la familia de grafos utilizada.

4.5. Otros

Más allá de los aspectos que se desprenden de alguna prueba en particular, hay otros que también resultan interesantes.

Esfuerzo: en todo el trabajo se mide como la cantidad de evaluaciones que insume la obtención de una solución aunque se mencionó que hay otros factores que podrían ser tenidos en cuenta. La decisión de adoptar esta medida tiene la ventaja de permitir una comparación sencilla entre estrategias, y la desventaja de esconder en su interior una parte del *tiempo* de cómputo insumido. Por ejemplo, aunque una instancia grande y otra pequeña requieran, ambas, una sola evaluación según una estrategia golosa, comúnmente la grande demorará más que la pequeña; el mismo fenómeno se observa en la disminución de la cantidad de evaluaciones de la Figura 15 (Límites de GRASP) donde, para evitar que aumente el tiempo total de cómputo, se restringe el análisis. El proceso de actualización del universo de candidatos, que se ejecuta luego de la selección propia de cada estrategia, es un contribuyente importante al tiempo total insumido, aunque compartido por todas las estrategias. Como alternativa a la cantidad de evaluaciones se tuvo en cuenta a la cantidad de comparaciones entre candidatos, pero se la descartó porque aportaba un nivel de detalle tan minucioso que en este trabajo sólo entorpece las explicaciones. De todas maneras esa u otras alternativas valen la pena ser tenidas en cuenta si se desea avanzar en la capacidad de predecir o disminuir el tiempo insumido.

Grafos aleatorios: la capacidad de los grafos aleatorios para representar a los hipergrafos promedio depende en gran medida de la manera en que se los construya y de qué se entienda por promedio; sin ningún tipo de consideraciones, la instancia promedio tendría un tamaño o cantidad de arcos exponencial, respecto de los nodos, que a su vez sería muy poco probable, luego poco interesante, para el contexto esperado de uso. En cambio, la manera en que se construyeron las instancias aleatorias de estas pruebas pretende resolver ese inconveniente mediante una función de crecimiento, que guía los valores de probabilidades involucrados en la generación de instancias. Que esta función de crecimiento logre instancias representativas de las esperadas es, como mínimo, optimista y, como mucho, ingenuo. Con más conocimiento sobre los casos esperados podrían generarse instancias que los representen; sin él, queda la esperanza de haber logrado una aproximación razonable.

Situaciones consideradas: si bien los grafos de estas pruebas aspiran a cubrir una cantidad grande de situaciones que se asumen esperables, indudablemente no alcanzan a la totalidad de las posibilidades; más temprano se citó un ejemplo referido a la ausencia de instancias con múltiples CFC. Podrán conseguirse nuevas instancias que cubran algunas de las situaciones vacantes... y seguirán quedando otras sin explorar. Se entendió que los casos estudiados alcanzaron a cumplir con su objetivo.

Pruebas en lote: el uso de la implementación para procesar lotes de instancias se aleja del caso de uso esperado, en el que una única instancia es analizada por vez. Así, lotes que demoraron horas o días en estudiarse, de todas maneras son útiles para conocer el comportamiento de la implementación

ya que, consideradas las instancias por separado y no el lote en su conjunto, generalmente se trata de elementos que pueden procesarse en segundos o minutos, dentro de grupos que pueden contener cientos de otras instancias. La demora exagerada en procesar un lote no invalida sus resultados.

Integración con *ReMo*: volviendo al comienzo, el fin último de este trabajo consiste en acelerar aquel verificador de modelos. En la interacción de ambas implementaciones, no obstante, existe una asimetría: el esfuerzo demandado al verificador para procesar una especificación es siempre potencialmente mucho mayor que el demandado a esta etapa de simplificación. Por lo tanto, instancias simplificables fácilmente pueden resultar inabordables para el verificador y no así a la inversa. ¿Qué ventaja se obtiene de simplificar instancias mayores a las que actualmente puede procesar este trabajo, si luego resultarán casi seguramente inabordables para el verificador? La respuesta no es evidente. Las causas de esa asimetría hacen prever que cualquier evolución del verificador, a lo sumo, disminuya la brecha y empareje los esfuerzos; recién entonces será conveniente retomar la preocupación por acelerar este trabajo.

5. Trabajo futuro

Esta sección propone continuaciones al presente trabajo. En algunos casos se trata de pequeños cambios a la implementación, en otros de la incorporación de nuevas ideas y, por último, algunos enfoques complementarios. De otras secciones sólo recojo las ideas más atractivas, sin desmerecer a las aristas sin pulir que puedan subsistir.

El primer grupo de cambios a considerar está formado por modificaciones sencillas que den respuesta a algunas asperezas. Una propuesta que abarca a todas las estrategias consiste en revisar y mejorar la implementación de `updateCandidates(...)`; actualmente realiza bastante trabajo innecesario, aumentando la incidencia de factores ajenos a la cantidad de evaluaciones en el tiempo total de cómputo. En cambio, una propuesta que sólo atañe a la estrategia GRASP consiste en estimar el total de esfuerzo que dedicará a resolver heurísticamente una instancia y, de superar lo estimado necesario para resolverla exhaustivamente, optar por el comportamiento que garantiza una solución exacta.

Un próximo grupo de cambios está constituido por ideas para nuevos preprocesos de optimización local, es decir, nuevos integrantes de la familia de «mejoras». Por ejemplo, podría refinarse la descomposición en subgrafos para analizar porciones más pequeñas que las actuales, o buscarse arcos y nodos en la intersección de varios subgrafos en los que la elección de candidatos pueda resolverse golosamente. Una atractiva fuente de inspiración se halla en las propiedades de los grafos reducibles cíclicamente (*cyclically reducible graphs*) según [WLS85]. Ensayos con pruebas de concepto arrojaron resultados prometedores; resta comprobar la validez de cada propuesta. Nuevas preguntas que habrá que contestar en esta dirección incluyen la efectividad de combinar distintas propuestas y hasta qué punto es indistinto el orden en que se aplica cada una.

Por último, también es posible prestar atención a alternativas más alejadas. Podría indagarse sobre la posibilidad de emplear algoritmos polinomiales de aproximación [Joh74] o adentrarse en otras manipulaciones simbólicas más allá de los reemplazos nacidos de ecuaciones simples.

Sin embargo, antes que sugerir cualquiera de estos caminos, el trabajo futuro recomendado consiste en someter la implementación a casos reales de estudio.

6. Conclusiones

Este trabajo se propuso desarrollar una herramienta en particular y cumplió con esa meta. En el camino se estudió la dificultad computacional del problema original, implementando y evaluando una familia de soluciones. A la vez, se continuó el trabajo del verificador de modelos ReMo, desarrollando una de las vías de continuación previstas en sus recomendaciones de trabajo futuro.

Se trabajó mucho para llegar a este punto, se descartaron montones de ideas, párrafos, líneas de código, horas de sueño y retazos de demostraciones; pero, más que nada, sirvió para abrir la puerta a una nueva etapa.

Referencias

[AS92] Noga Alon and Joel H. Spencer: *The probabilistic method*. Wiley, New York, 1992, ISBN 0-471-53588-5. `http://cs.nyu.edu/cs/faculty/spencer/nogabook/nogabook.html`.

[Bol98] Béla Bollobás: *Modern graph theory*. Springer-Verlag, 1998, ISBN 0-387-98488-7.

[CGP00] Edmund M. Clarke, Orma Grumberg, and Doron Peled: *Model checking*. MIT Press, January 2000, ISBN 0-262-03270-8.

[Die00] Reinhard Diestel: *Graph theory*. Springer-Verlag, 2nd edition, 2000. `http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryII.pdf`.

[DP01] Nachum Dershowitz and David A. Plaisted: *Rewriting*. In John Alan Robinson and Andrei Voronkov (editors): *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001, ISBN 0-444-50813-9; 0-262-18223-8. `http://citeseer.ist.psu.edu/455341.html`.

[ER60] Paul Erdös and Alfréd Rényi: *On the evolution of random graphs*. Publications of the Mathematical Institute of the Hungarian Academy of Sciences, 5:17–61, 1960.

[Fag83] Ronald Fagin: *Degrees of acyclicity for hypergraphs and relational database schemes*. Journal of the ACM, 30(3):514–550, July 1983.

[FM95] Jeremy Frank and Charles U. Martel: *Phase transitions in the properties of random graphs*, June 24 1995. `http://ic.arc.nasa.gov/people/frank/hamphase.cp95.ps`.

[FR89] Tomas A. Feo and Mauricio G. C. Resende: *A probabilistic heuristic for a computationally difficult set covering problem*. Operations Research Letters, 8:67–71, 1989. `http://public.research.att.com/~mgcr/doc/gsc.pdf`.

[FR95] Tomas A. Feo and Mauricio G. C. Resende: *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6:109–133, 1995. `http://public.research.att.com/~mgcr/doc/gtut.pdf`.

[Fri02] Marcelo Fabián Frias: *Fork Algebras in Algebra, Logic and Computer Science*. World Scientific Publishing Co., Inc., 2002.

[GJ79] Michael R. Garey and David S. Johnson: *Computers and Intractability*. W. H. Freeman & Co., 1979, ISBN 0-716-71045-5.

[GJM02] Carlo Ghezzi, Mehdi Jazayeri, and D. Mandrioli: *Fundamentals of software engineering*. Prentice Hall, second edition, 2002.

[GLNP93] Gallo, Longo, Nguyen, and Pallottino: *Directed hypergraphs and applications*. Discrete Applied Mathematics, 42(2):177–201, 1993. `http://citeseer.ist.psu.edu/71942.html`.

[Glo86] Fred Glover: *Future paths for integer programming and links to artificial intelligence.* Computers and Operations Research, 13(5):533–549, 1986.

[Glo89] Fred Glover: *Tabu search–Part I.* ORSA Journal on Computing, 1(3):190–206, 1989.

[GS05] Rodolfo Gamarra y Gabriela Steren: *Implementación de una herramienta de model-checking basada en álgebra relacional.* Tesis de Licenciatura, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, UBA, 2005. Director: Marcelo Frias.

[Har69] Frank Harary: *Graph Theory.* Addison-Wesley, 7th printing edition, 1969, ISBN 0-201-02787-9; 0-201-41033-8.

[HT73] John Hopcroft and Robert Tarjan: *Algorithm 447: efficient algorithms for graph manipulation.* Communications of the ACM, 16(6):372–378, 1973, ISSN 0001-0782.

[Joh74] David S. Johnson: *Approximation algorithms for combinatorial problems.* Journal of Computer and System Sciences, 9(3):256–278, December 1974.

[KCDGV83] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi: *Optimization by simulated annealing.* Science, 220(4598):671–680, 1983.

[Klo92] Jan Willem Klop: *Term rewriting systems.* In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum (editors): *Handbook of Logic in Computer Science, Clarendon*, volume II, chapter 1, pages 1–117. Oxford University Press, 1992. `http://citeseer.ist.psu.edu/klop92term.html`.

[Koz95] John R. Koza: *Survey of genetic algorithms and genetic programming.* In *Proceedings of 1995 WESCON Conference*, pages 589–594. IEEE, 1995.

[NSS94] Esko Nuutila and Eljas Soisalon-Soininen: *On finding the strongly connected components in a directed graph.* Information Processing Letters, 49(1):9–14, 1994, ISSN 0020-0190. `http://www.cs.hut.fi/~enu/ps/ipl-scc.ps`.

[Res99] Mauricio G. C. Resende: *Greedy randomized adaptive search procedures (GRASP)*, 1999. `http://public.research.att.com/~mgcr/doc/sgrasp.pdf`.

[RPP00] Mauricio G. C. Resende, Leonidas S. Pitsoulis, and Panos M. Pardalos: *Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP.* Discrete Applied Mathematics, 100:95–113, 2000.

[WLS85] Ching Chy Wang, Errol L. Lloyd, and Lou Soffa: *Feedback vertex sets and cyclically reducible graphs.* Journal of the ACM, 32(2):296–313, April 1985.

A. Complemento técnico

Este apéndice profundiza en las definiciones y demostraciones del Desarrollo.

A.1. Candidatos

A continuación se presentan los conceptos que permiten definir con rigor la noción de *candidato* y que más adelante se emplearán en demostraciones que los involucran.

Definición 34 (alcanzabilidad)**:** dado $H =< V, X >$ un hipergrafo se dice que un nodo v es *alcanzable* desde otro nodo u en el grafo si, y sólo si, existe un camino en H que comienza con u y termina en v. Se nota $u \xrightarrow{C} v$ para indicar que v es alcanzable desde u mediante arcos en $C \subseteq X$.

Definición 35 (frontera)**:** dados $H =< V, X >$ un g.s., $C = \{e_1, \ldots, e_n\} \subseteq X$ una colección determinística y $e = (o_e, D_e) \in X$, $F \subseteq V$ se dice la *frontera* de nodos alcanzables desde «e» mediante C y se escribe $e \xrightarrow{C} F$ para indicar que $F = \{v \in V : o_e \overset{C\cup\{e\}}{\rightarrow} v \text{ y } v \neq origen(e_i) \text{ para } 1 \leq i \leq n\}$. Además, si $v \in F$ se escribe $e \xrightarrow{C} v$. Si bien puede resultar poco intuitivo basar la definición sobre un arco, partir de un nodo obligaría a declarar de todos modos un arco distinguido, con el agravante de que tener que añadir la restricción de que su origen coincida con el nodo. Hecha la advertencia, se consideró mejor la alternativa actual.

Definición 36 ($tail$)**:** dada $L = [e_1, e_2, \ldots, e_n]$, $tail(L) = [e_2, \ldots, e_n]$ si $n > 0$.

Definición 37 ($L\sigma$)**:** dada $L = [e_1, e_2, \ldots, e_n]$, si tiene sentido aplicar una sustitución a sus elementos y σ es una sustitución, $L\sigma = [e_1\sigma, \ldots, e_n\sigma]$.

Definición 38 (Q^j, L^j y σ^j)**:** dados Q un sistema de ecuaciones y L una lista $[\sigma_1, \ldots, \sigma_n]$ de Q-sustituciones escribo $L^0 = L$ y, si $\sigma \in Q_{\rightarrow}$, $\sigma^{L^0} = \sigma$; si además existen L^j y σ^{L^j} y el primer elemento de L^j es $\sigma_{p_j}^{L^j}$ entonces, si es una sustitución, escribo $\sigma^{L^{j+1}} = \sigma^{L^j}\sigma_{p_j}^{L^j}$ y, si sus elementos están definidos, $L^{j+1} = tail(L^j)\sigma_{p_j}^{L^j}$. También, $Q^{L^0} = Q$ y, si existen Q^{L^j} y $\sigma_{p_j}^{L^j}$, $Q^{L^{j+1}} = Q^{L^j}\sigma_{p_j}^{L^j}$. La extensión a especificaciones vale con igual definición que en los sistemas de ecuaciones y, si no es ambiguo, puedo omitir la mención a la lista para aligerar la lectura. Observar que, si existen, $reemplazar(L^j, Q^j) = reemplazar(L^{j+1}, Q^{j+1})$, $L^n = [\ ]$ y $L^j = [\sigma_{p_j}^j, \ldots, \sigma_n^j]$. Además, si L es Q-compatible, vale que $p_j = j + 1$.

En un acercamiento progresivo a la definición de *candidato*, el primer paso es conocer la intuición. En cada momento del proceso de minimización de especificaciones, un candidato representa un elemento que conduce a nuevas soluciones. Según se dejó entender más temprano, tiene un carácter doble: vinculado con la especificación original, encarna una sustitución en el sistema de ecuaciones asociado; vinculado con la solución parcial vigente, encarna un hiperarco del g.s. construido sobre la representación asociada, producto de aplicar consecutivamente los reemplazos en la solución parcial a la especificación original.

Definición 39 (candidato elemental)**:** dado Q un sistema de ecuaciones, un *candidato elemental* para Q es un par ordenado $c = (\sigma_f, e)$ tal que $f \in Q$ y $e = rep(\sigma_f)$. Se extiende análogamente la definición para las especificaciones.

Definición 40 (candidato)**:** dados Q un sistema de ecuaciones y L una lista $[\sigma_1, \ldots, \sigma_k]$ de Q-sustituciones, si existe σ^{L^k}, un *candidato* es un par ordenado $c^{L^k} = (\sigma, e^{L^k})$ tal que $\sigma \in Q_{\rightarrow}$ y $e^{L^k} = rep(\sigma^{L^k})$. Pueden omitirse la lista o el superíndice cuando no resulte ambiguo; también, se extiende la definición para las especificaciones.

Afirmación 3: dados Q un sistema de ecuaciones, $H = rep(Q)$ un g.s., una lista $L = [\sigma_1, \ldots, \sigma_k]$ de Q-sustituciones, una colección $C = \{e_1, \ldots, e_k\}$ con $e_i = rep(\sigma_i)$ para $1 \leq i \leq k$ y $c^{L^k} = (\sigma, e^{L^k})$ un candidato, si $e^{L^k} = (o_e, D_{e^k})$ entonces $rep(\sigma) \xrightarrow{C} D_{e^k}$.

Demostración: hay que considerar los casos: 1) $k = 0$ y 2) $k > 0$.

1) $C = \emptyset$ y $rep(\sigma) \xrightarrow{\emptyset} D_{e^0}$ cumple por definición.

2) $C \neq \emptyset$, $rep(\sigma) \xrightarrow{C_j} F_j$ si $C_j = \{e_1, \ldots, e_j\}$ para $1 \leq j \leq k$ y, cuando $j \neq k$, se asume que $F_j = D_{e^j}$. Dadas *consts* la función que dado un término devuelve el conjunto de los símbolos de constante que aparecen en él y, si $e = (o_e, D_e)$ es un arco, $dest(e) = D_e$, quiero ver que valen 2.1) $F_k \subseteq D_{e^k}$ y 2.2) $D_{e^k} \subseteq F_k$.

2.1) Quiero ver que $v \in F_k \Rightarrow v \in D_{e^k}$ considerando:

2.1.1) caso $v \in F_{k-1}$

$v \in F_k$	$\Rightarrow$	$v \neq izq(\sigma_k)$	(Definición 35)
$v \in F_{k-1}$	$\Rightarrow$	$v \in D_{e^{k-1}}$	(hipótesis)
	$\Rightarrow$	$v \in dest(rep(\sigma^{L^{k-1}}))$	(Definición 40)
	$\Rightarrow$	$v \in consts(der(\sigma^{L^{k-1}}))$	(Definición 21)
	$\Rightarrow$	$v \in consts(der(\sigma^{L^{k-1}}\sigma_k^{k-1}))$	$(v \neq izq(\sigma_k))$
	$\Rightarrow$	$v \in consts(der(\sigma^{L^k}))$	(Definición 38)
	$\Rightarrow$	$v \in dest(rep(\sigma^{L^k}))$	(Definición 21)
	$\Rightarrow$	$v \in D_{e^k}$	(Definición 40)

2.1.2) caso $v \notin F_{k-1}$

$v \notin F_{k-1}$	$\Rightarrow$	$v \in consts(der(\sigma_k^{L^{k-1}}))$	(hipótesis)
$v \notin F_{k-1}$	$\Rightarrow$	$izq(\sigma_k) \in F_{k-1}$	(Definición 34)
	$\Rightarrow$	$izq(\sigma_k) \in D_{e^{k-1}}$	(hipótesis)
	$\Rightarrow$	$izq(\sigma_k) \in dest(rep(\sigma^{L^{k-1}}))$	(Definición 40)
	$\Rightarrow$	$izq(\sigma_k) \in const(der(\sigma^{L^{k-1}}))$	(Definición 21)
	$\Rightarrow$	$v \in const(der(\sigma^{L^{k-1}}\sigma_k^{k-1}))$	$(v \in consts(der(\sigma_k^{L^{k-1}})))$
	$\Rightarrow$	$v \in const(der(\sigma^{L^k}))$	(Definición 38)
	$\Rightarrow$	$v \in dest(rep(\sigma^{L^k}))$	(Definición 21)
	$\Rightarrow$	$v \in D_{e^k}$	(Definición 40)

De 2.1.1) y 2.1.2) se obtiene que vale 2.1), es decir, $F_k \subseteq D_{e^k}$.

2.2) Quiero ver que $v \in D_{e^k} \Rightarrow v \in F_k$ considerando:

2.2.1) caso $v \in D_{e^{k-1}}$

$$
\begin{array}{rcll}
v \in D_{e^k} & \Rightarrow & v \neq izq(\sigma_k) & \text{(Definición 35)} \\
v \in D_{e^{k-1}} & \Rightarrow & v \in F_{k-1} & \text{(hipótesis)} \\
 & \Rightarrow & rep(\sigma) \overset{C_{k-1}}{\rightharpoonup} v & \text{(definición de } F_{k-1}\text{)} \\
 & \Rightarrow & izq(\sigma) \overset{C_{k-1} \cup \{rep(\sigma)\}}{\longrightarrow} v & \text{(Definición 35)} \\
 & \Rightarrow & izq(\sigma) \overset{C_{k-1} \cup \{e_k, rep(\sigma)\}}{\longrightarrow} v & \text{(Definición 34)} \\
 & \Rightarrow & izq(\sigma) \overset{C_k \cup \{rep(\sigma)\}}{\longrightarrow} v & \text{(definición de } C_k\text{)} \\
 & \Rightarrow & rep(\sigma) \overset{C_k}{\rightharpoonup} v & (v \neq izq(\sigma_k)) \\
 & \Rightarrow & v \in F_k & \text{(definición de } F_k\text{)}
\end{array}
$$

2.2.2) caso $v \notin D_{e^{k-1}}$

$$
\begin{array}{rcll}
v \notin D_{e^{k-1}} & \Rightarrow & v \in consts(der(\sigma_k)) & \text{(hipótesis)} \\
v \notin D_{e^{k-1}} & \Rightarrow & izq(\sigma_k) \in D_{e^{k-1}} & \text{(Definición 34)} \\
 & \Rightarrow & izq(\sigma_k) \in F_{k-1} & \text{(hipótesis)} \\
 & \Rightarrow & rep(\sigma) \overset{C_{k-1}}{\rightharpoonup} izq(\sigma_k) & \text{(definición de } F_{k-1}\text{)} \\
 & \Rightarrow & izq(\sigma) \overset{C_{k-1} \cup \{rep(\sigma)\}}{\longrightarrow} izq(\sigma_k) & \text{(Definición 35)} \\
 & \Rightarrow & izq(\sigma) \overset{C_{k-1} \cup \{e_k, rep(\sigma)\}}{\longrightarrow} izq(\sigma_k) & \text{(Definición 34)} \\
 & \Rightarrow & izq(\sigma) \overset{C_k \cup \{rep(\sigma)\}}{\longrightarrow} izq(\sigma_k) & \text{(defínción de } C_k\text{)} \\
 & \Rightarrow & izq(\sigma) \overset{C_k \cup \{rep(\sigma)\}}{\longrightarrow} v & (v \in consts(der(\sigma_k))) \\
 & \Rightarrow & rep(\sigma) \overset{C_k}{\rightharpoonup} v & (v \neq v_k) \\
 & \Rightarrow & v \in F_k & \text{(definición de } F_k\text{)}
\end{array}
$$

De 2.2.1) y 2.2.2) se obtiene que vale 2.2), es decir, $D_{e^k} \subseteq F_k$. Luego, como valen 2.1) y 2.2), vale 2), es decir, si $k > 0$, $D_{e^k} = F_k$. □

A.2. Compatibilidad y determinismo

A continuación se establece la estrecha relación que existe entre las nociones de *compatibilidad* y *determinismo*.

Afirmación 4: sean Q un sistema de ecuaciones, $H =< V, X >$ un g.s. tal que $H = rep(Q)$, $S = [\sigma_1, \ldots, \sigma_n]$ una lista de Q-sustituciones y $Z = [e_1, \ldots, e_n]$ con $e_i = rep(\sigma_i)$ para $1 \leq i \leq n$, entonces S es compatible si, y sólo si, Z es determinística.

Demostración: conviene partir de las condiciones planteadas y considerar dos casos separados:

1. Z determinística $\Rightarrow$ S compatible;
2. S compatible $\Rightarrow$ Z determinística.

1) caso Z determinística $\Rightarrow$ S compatible

Probar que $S = S^0$ sea compatible requiere probar que S^0 sea una lista de Q^0-sustituciones y que, si existe, S^1 sea Q^1-compatible. Suponiendo a S^0 una lista de Q^0-sustituciones se vuelve a un punto similar al de partida, ya que en lugar de probar a S^0 Q^0-compatible se debiera probar a S^1 Q^1-compatible.

Generalizando este razonamiento, si se asume dada S^j una lista cuyos elementos son Q^j-sustituciones, probar su Q^j-compatibilidad se reduce a probar,

si existe, la Q^{j+1}-compatibilidad de S^{j+1}; si alguna lista fuese vacía el análisis sería sencillo. La utilidad de esta reducción se halla en que las listas son finitas, la longitud de S^{j+1}, si existe, es estrictamente menor que la de S^j y en la fácil determinación de la compatibilidad de una lista vacía; es decir, en la existencia de un $k \leq n$ tal que, si es una lista, $S^k = [\ ]$ garantice la terminación de la prueba. Esquemáticamente:

$$\begin{aligned}
S^0\ Q^0\text{-compatible} &\equiv S^0\ Q^0\text{-sustituciones y } S^1\ Q^1\text{-compatible} \\
S^1\ Q^1\text{-compatible} &\equiv S^1\ Q^1\text{-sustituciones y } S^2\ Q^2\text{-compatible} \\
&\vdots \\
S^{k-1}\ Q^{k-1}\text{-compatible} &\equiv S^{k-1}\ Q^{k-1}\text{-sustituciones y } S^k\ Q^k\text{-compatible} \\
S^k = [\ ]\ Q^k\text{-compatible} &\equiv \textit{verdadero} \text{ (por definición)}
\end{aligned}$$

Llegado este punto, la demostración se reduce a probar que, para $j \leq k$, S^j sea una lista de Q^{S^j}-sustituciones. Como se explicó en la Definición 9 (lista compatible), la dificultad del caso radica en justificar la existencia de tales listas. Esta demostración se completa con la Afirmación 5, que más adelante se prueba verdadera. En tal situación es inmediato que S es compatible.

2) caso S compatible $\Rightarrow$ Z determinística

Probar esta implicación no difiere significativamente del caso anterior. □

Definición 41 (prefijo)**:** dada $L = [a_1, \ldots, a_n]$ una lista, su prefijo $[a_1, \ldots, a_k]$ con $k \leq n$ se escribe L_k.

Afirmación 5: en iguales condiciones que la Afirmación 4 y suponiendo que Z es determinística, para $j \leq n$ existen las listas S^j de Q^{S^j}-sustituciones.

Demostración: con $Q = Q^{S^0}$ un sistema de ecuaciones, $H =< V, X >$ un g.s. tal que $H = rep(Q)$, $S = [\sigma_1, \ldots, \sigma_n]$ una lista de Q-sustituciones, $e_x^y = rep(\sigma_x^y)$ y $Z = [e_1, \ldots, e_n]$ una lista determinística, ya que $S = S^0$ existe por hipótesis alcanza con garantizar que, si $j < n$ y S^j es una lista de Q^j-sustituciones, S^{j+1} sea una lista de Q^{j+1}-sustituciones.

Con S^j una lista de Q^j-sustituciones, para todo σ_i^j en S^j debe existir una f.s. f_i^j en Q^j tal que σ_i^j sea $\sigma_{f_i^j}$. Si S^j es vacía se completa la demostración; si no, existe un primer elemento $\sigma_{p_j}^j$ de S^j. Sin pérdida de generalidad, si f_i^j es la fórmula $c_i = t_i^j(x_1, \ldots, x_m)$, por la Definición 4 (reemplazo), $f_i^j \sigma_{p_j}^j$ es $c_i \sigma_{p_j}^j = t_i^j(x_1, \ldots, x_m)\sigma_{p_j}^j$ y se quiere saber si esta última ecuación es en sí una f.s. cuando $i \neq p_j$ o, en otras palabras, si da origen a una nueva sustitución. Conviene analizar cada miembro de la ecuación por separado.

Respecto de $c_i \sigma_{p_j}^j$: $i \neq p_j \Rightarrow \sigma_i^j \neq \sigma_{p_j}^j$ y Z determinística implica que $d_{out}^Z(c_i) \leq 1$, luego $c_i \neq izq(\sigma_{p_j}^j)$. Se sigue que $c_i \sigma_{p_j}^j = c_i$.

Respecto de $t_i^j(x_1, \ldots, x_m)\sigma_{p_j}^j$: ya que por definición el término es igual a $t_i^j(x_1\sigma_{p_j}^j, \ldots, x_m\sigma_{p_j}^j)$ alcanza con estudiar $x_k\sigma_{p_j}^j$ para $1 \leq k \leq m$. Pretendo mostrar que c_i no puede aparecer en esas expresiones. Razonando por absurdo

supongo que sí figura; entonces, para algún k_0 deberá suceder que $\sigma^j_{p_j}$ es la sustitución $x_{k_0} \to t^j_{p_j}(c_i, y_1, \ldots, y_{\tilde{n}})$.

Notando $r = p_j$, por hipótesis existen el prefijo $P = S^j_j$ de S^j y $\sigma^P_r = \sigma^j_{p_j}$; luego, existe el candidato $c^P_r = (\sigma_r, e^P_r)$ con $e^P_r = rep(\sigma^P_r) = (x_{k_0}, D_{e^P_r})$ y $D_{e^P_r} = \{c_i, y_1, \ldots, y_{\tilde{n}}\}$. Entonces, $e_r = rep(\sigma_r) \overset{Z_j}{\rightharpoonup} D_{e^P_r}$ por la Afirmación 3 y, por lo tanto, $e_r \overset{Z_j}{\rightharpoonup} c_i$ y $x_{k_0} \overset{Z_j \cup \{e_r\}}{\to} c_i$. Con el mismo argumento, gracias a σ^j_i se sabe que $c_i \overset{Z_j}{\to} x_{k_0}$. Como consecuencia debe valer $c_i \overset{Z}{\to} c_i$, lo que contradice que Z sea determinística.

Se sigue que c_i no aparece en $x_k \sigma^j_{p_j}$ para $1 \leq k \leq m$, $\sigma^j_i \sigma^j_{p_j} = \sigma^{j+1}_i$ existe si $i \neq p_j$ y, finalmente, S^{j+1} es una lista de Q^{j+1}-sustituciones. □.

La validez de la Afirmación 1 en el Desarrollo está justificada por la Afirmación 4, asumiendo $Q = E_=$ y $ap(rep(\sigma_i)) = \sigma_i$ para $1 \leq i \leq n$.

A.3. Soluciones viables

A continuación se argumenta la validez de las soluciones parciales y se completa la justificación del procedimiento presentado en la Sección 2.4.3.

Afirmación 6: dados Q un sistema de ecuaciones, $L = [\sigma_1, \ldots, \sigma_k]$ una lista Q-compatible y $c^{L^k} = (\sigma, e^{L^k})$ un candidato, $L' = [\sigma_1, \ldots, \sigma_k, \sigma]$ es una nueva lista Q-compatible.

Demostración: si $e_i = rep(\sigma_i)$, $C = [e_1, \ldots, e_k]$, $S+T$ es la concatenación de dos listas y $C' = C + [rep(\sigma)]$, por la Afirmación 4 se sabe que L' es compatible si, y sólo si, C' es determinística. Se debe ver que: 1) $e = (o_e, D_e) \in C' \Rightarrow d^{C'}_{out}(o_e) \leq 1$ y 2) C' es acíclica.

1) Que $d^C_{out}(o_{e_i}) \leq 1$ se sabe para $1 \leq i \leq k$ porque C es determinística; resta ver que además $izq(\sigma) \neq izq(\sigma_i)$. Por absurdo, si σ_{i_0} es tal que $izq(\sigma) = izq(\sigma_{i_0})$, no estaría definida $\sigma^{L^{i_0}} = \sigma^{L^{i_0-1}} \sigma^{i_0-1}_{i_0}$ ya que los reemplazos en sustituciones requieren partes izquierdas distintas; luego, tampoco estaría definida σ^{L^k}, condición necesaria para la existencia de $e^{L^k} = rep(\sigma^{L^k})$ en c^{L^k}. Se sigue que $d^{C'}_{out}(o_e) \leq 1$ si $e \in C'$.

2) De haber un ciclo en C' es necesario que $rep(\sigma)$ forme parte de él, ya que C era acíclica. En ese caso, si $rep(\sigma^y_x) = (o_{\sigma^y_x}, D_{\sigma^y_x})$ deben existir σ_{i_0} y σ_{i_1} en L tales que $o_{\sigma_{i_0}} \in D_\sigma$, $o_\sigma \in D_{\sigma_{i_1}}$ y $o_{\sigma_{i_0}} \overset{C}{\to} o_{\sigma_{i_1}}$, lo que junto a 1) implican que $rep(\sigma) \overset{C}{\rightharpoonup} o_\sigma$ y, por la Afirmación 3, $o_\sigma \in D_{\sigma^{L^k}}$. Un absurdo ya que en ese caso e^{L^k} es un arco inválido para un g.s. y c^{L^k} resulta un candidato inválido. Se sigue que C' es acíclica. □

Afirmación 7: el procedimiento descrito en la Sección 2.4.3 para la actualización del universo de candidatos es correcto.

Demostración: ya que la Afirmación 6 garantiza que los candidatos conduzcan siempre a soluciones viables, resta justificar que el procedimiento realiza tanto una actualización necesaria como suficiente. Que sea necesaria se explicó en

2.4.3, con la separación en casos; que sea suficiente se desprende de la Definición 40 (candidato) y de la Afirmación 3: como el destino del hiperarco en los candidatos es igual a la frontera de nodos alcanzables mediante la solución parcial, que a su vez coincide por construcción con el procedimiento de actualización, el resultado es una nueva colección de candidatos, con todos los cambios necesarios. □

A.4. Hojas

A continuación se completa la justificación del argumento expuesto en la Sección 2.6.3.

Definición 42 (conservación)**:** dados $H =< V, X >$ un g.s. y $C \subseteq X$ determinística, $X_C = \{e \in X : \{e\} \cup C$ es determinística pero $e \notin C\}$ y $V_C \subseteq V$ es el conjunto de nodos tales que algún arco en X_C los tiene por origen. Se dice que C *conserva* a V_C y X_C en H.

La intuición de fondo es que si C fuera una solución parcial al problema de optimización que se quiere resolver, tras realizar reemplazos inducidos por C en una especificación, permanecerían V_C y reescrituras de las ecuaciones que dan origen a X_C en la representación del resultado.

Afirmación 8: en un g.s. acíclico toda solución maximal conserva únicamente a las hojas del grafo.

Demostración: dados $H =< V, X >$ un g.s. y $Z = [e_1, \ldots, e_n]$ una solución maximal para H, por absurdo, supongo que $v \in V$ no es hoja y $v \in V_Z$. Entonces debe existir $e = (v, D_e) \in X = X_{Z_0}$ con $D_e \subset V$ y, dado que Z es maximal, $e \notin X_Z = X_{Z_n} = \emptyset$; luego, existe $1 \leq i \leq n$ tal que $e \in X_{Z_{i-1}}$ pero $e \notin X_{Z_i}$. Por definición, los motivos posibles son:

- que $d_{out}^{\{e\} \cup Z_i}(v) > 1$; no puede suceder ya que en este caso el origen de e_i debe ser v y $e_i \in \overline{X_{Z_i}}$ se contradice con $v \in V_{Z_i}$, luego con $v \in V_Z$.
- que $\{e\} \cup Z_i$ sea conflictiva; no puede suceder ya que la existencia de ciclos se contradice con los supuestos iniciales.

Como el absurdo provino de suponer la existencia de v, se concluye que toda solución maximal conserva únicamente a las hojas del grafo. □

B. Etiquetas

El model-checker con que se integra este trabajo considera distintos tipos de relaciones binarias; dados dos conjuntos, una *etiqueta* identifica el tipo específico. Si A y B son conjuntos y $\bullet$ una etiqueta, el tipo de la relación $A \bullet B$ se rige por el Cuadro 4.

etiqueta	relación
`*`	sin restricciones
`->`	función parcial
`:->`	función total
`=>`	función constante
`.`	átomo

Cuadro 4: Etiquetas del model-checker

En cada caso se calcula la cantidad de posibles instancias del tipo y se asigna ese valor al peso de la relación. Por ejemplo, si R es un símbolo de constante tal que $R \in A * B$, $|A| = n$ y $|B| = m$, $peso(R) = 2^{n \cdot m}$.

A los tipos de relaciones que soporta el model-checker se agregaron nuevas funciones más específicas, detalladas en el Cuadro 5.

etiqueta	relación
`-i>`	inyectiva parcial
`-s>`	sobreyectiva parcial
`-b>`	biyectiva parcial
`:-i>`	inyectiva total
`:-s>`	sobreyectiva total
`:-b>`	biyectiva total

Cuadro 5: Etiquetas nuevas

C. Implementación

Este apéndice detalla algunos aspectos salientes de la implementación.

C.1. Notas

A continuación se exponen los detalles que por algún motivo revisten de interés técnico. A saber:

- Las tres estrategias implementadas ordenan al universo de candidatos en una lista, de mayor a menor según su peso. En su ejecución, más pesado primero se limita a tomar siempre el primer elemento de la lista vigente, sin realizar ningún tipo de búsqueda, mientras que GRASP logra evitarse construir la lista restringida de candidatos considerando un prefijo adecuado de la misma lista; por lo tanto, las tres estrategias son capaces de seleccionar candidatos con esfuerzo constante respecto de la cantidad de elementos.
- La implementación de GRASP dispone de dos parámetros de configuración: el tiempo durante el cual le está permitido ejecutar a una iteración, y la cantidad de iteraciones que se desean. Predeterminadamente se correrán 128 iteraciones de 1 (un) segundo cada una.
- Mediante el uso de un cronómetro, la implementación de GRASP es independiente de la velocidad de la computadora en que se ejecuta.

C.2. Uso

Para ejecutar el programa se escribe:

```
minime.Minime [opciones] espec1 [espec2 ... especN]
```

donde cada `espec` es una especificación en el lenguaje que acepta *ReMo* y las opciones son:

`--brute` emplear estrategia de fuerza bruta ($\star$);

`--heavy` emplear estrategia más pesado primero;

`--grasp` emplear estrategia GRASP;

`--daghCompression` habilitar compresión de dahgs ($\star$);

`--noDaghCompression` deshabilitar compresión de dahgs.

Las ($\star$) marcan las opciones predeterminadas.