ANÁLISIS E IMPLEMENTACIÓN DE MECANISMOS DE RESTRICCIONES DÉBILES PARA PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES

ALBERTO DELGADO ORTEGÓN
JORGE ANDRÉS PÉREZ PARRA

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
SANTIAGO DE CALI
2005

ANÁLISIS E IMPLEMENTACIÓN DE MECANISMOS DE RESTRICCIONES DÉBILES PARA PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES

ALBERTO DELGADO ORTEGÓN
JORGE ANDRÉS PÉREZ PARRA

Tesis de grado para optar al título de
Ingeniero de Sistemas y Computación

Director
CAMILO RUEDA CALDERÓN
Ingeniero de Sistemas y Computación

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
SANTIAGO DE CALI
2005

ARTICULO 23 de la Resolución No. 13 del 6 de Julio de 1946
del Reglamento de la Pontificia Universidad Javeriana.

"La Universidad no se hace responsable por los conceptos emitidos por sus alumnos en sus trabajos de tesis. Sólo velará porque no se publique nada contrario al dogma y a la moral católica y porque las tesis no contengan ataques o polémicas puramente personales; antes bien, se vea en ellas el anhelo de buscar la Verdad y la Justicia"

Santiago de Cali, Noviembre 8 de 2005

Doctor
JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería
Pontificia Universidad Javeriana
Ciudad

Certifico que el presente trabajo de grado, titulado "ANÁLISIS E IMPLEMENTACIÓN DE MECANISMOS DE RESTRICCIONES DÉBILES PARA PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES" realizado por ALBERTO DELGADO ORTEGÓN y JORGE ANDRÉS PÉREZ PARRA, estudiantes de Ingeniería de Sistemas y Computación, se encuentra terminado y puede ser presentado para sustentación.

Atentamente,

CAMILO RUEDA CALDERÓN
Director del Proyecto

Santiago de Cali, Noviembre 8 de 2005

Doctor
JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería
Pontificia Universidad Javeriana
Ciudad

Por medio de la presente presentamos a usted el trabajo de grado titulado "ANÁLISIS E IMPLEMENTACIÓN DE MECANISMOS DE RESTRICCIONES DÉBILES PARA PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES" para optar al título de Ingeniero de Sistemas y Computación.

Esperamos que este trabajo reúna todos los requisitos académicos y cumpla el propósito para el cual fue desarrollado, y sirva de apoyo para futuros proyectos en la Universidad Javeriana relacionados con la materia.

Atentamente,

Alberto Delgado O.
ALBERTO DELGADO ORTEGÓN

Jorge Andrés Pérez Parra
JORGE ANDRÉS PÉREZ PARRA

Nota de Aceptación:

Aprobado por el comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar al título de Ingeniero de Sistemas y Computación.

Dr. JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería

MARÍA CONSTANZA PABÓN
Directora de la Carrera de Ingeniería de Sistemas y Computación

CAMILO RUEDA CALDERÓN
Director de Tesis

RAFAEL JORDAN
Jurado

CARLOS ALBERTO OLARTE
Jurado

Agradecimientos

Los autores expresan sus agradecimientos:

- A Camilo Rueda, quien nos dío la oportunidad de trabajar en CRISOL.
- A Gustavo Gutiérrez, quien fue fundamental para el desarrollo de este trabajo. Gustavo no sólo colaboro desinteresadamente en la solución de múltiples inconvenientes de orden práctico, sino que también puso a nuestro servicio su experiencia para la presentación elegante de los resultados teóricos.
- Luis Omar Quesada colaboró con la lectura de algunos de los artículos científicos asociados con este trabajo, y sugirió detalles para complementar los análisis presentados en el capítulo 6.
- A Gustavo Gutiérrez, Julian Gutiérrez, Luis Omar Quesada y Carlos Ruiz por leer versiones iniciales de este documento y dar comentarios y sugerencias. Como es natural, cualquier error que permanezca es responsabilidad exclusiva de los autores.
- A todas aquellas personas que de una u otra forma colaboraron en la realización del presente trabajo.

La primera parte de este trabajo de grado fue desarrollado por los autores en el contexto de CRISOL (Constraint Research for Innovation in Software Solutions), proyecto adelantado por el Grupo de Investigación AVISPA entre Diciembre de 2002 y Febrero de 2005. CRISOL fue cofinanciado por COLCIENCIAS (Contrato No. 298-2002).

Adicionalmente, una segunda étapa del trabajo fue parcialmente financiada con fondos del proyecto de investigación "Desarrollo y Soporte de Contribuciones para Mozart", auspiciado por la Facultad de Ingeniería de la Pontificia Universidad Javeriana Cali entre Enero y Noviembre de 2005.

Índice general

Índice de cuadros

Índice de figuras

Lista de Algoritmos

Resumen

Diversos problemas en la vida real pueden modelarse como *problemas de satisfacción de restricciones* (o CSPs). En áreas como planeación resulta natural establecer las condiciones de un problema como *restricciones* sobre un conjunto de recursos disponibles. No obstante, esta formulación *clásica* de los CSPs resulta insuficiente para representar diversas situaciones de la vida real, en donde criterios como niveles de preferencias o costos asociados con las restricciones deben representarse adecuadamente, o en escenarios donde la naturaleza contradictoria de las condiciones del problema hace imposible encontrar una solución, por ejemplo. Para abordar esta problemática, los denominados modelos de *restricciones débiles* definen tipos especiales de CSPs en donde la solución puede no satisfacer algunas restricciones del problema original o puede satisfacerlas de forma incompleta (es decir, algún grado de violación es admitido). Más específicamente, algunos trabajos proponen formalismos *genéricos* que son capaces de representar diversos tipos de CSPs, lo que aumenta la aplicabilidad de las técnicas de restricciones débiles con respecto a tipos específicos de CSPs débiles.

Este trabajo de grado se enfoca en uno de estos formalismos genéricos de restricciones débiles, cuyo modelo está basado en una estructura matemática denominada *semianillo*, y pretende la implementación de mecanismos que permitan su uso en el contexto de Mozart, un lenguaje de programación *concurrente* por restricciones. Se espera que la integración armónica de dichos mecanismos con el modelo concurrente de Mozart permita el uso de restricciones débiles en la solución de problemas de la vida real.

Para lograr lo anterior se siguieron dos direcciones: en la primera se implementa un conjunto de restricciones débiles para Mozart a partir de una modificación conservativa al modelo basado en semianillos. Estas restricciones —implementadas como agentes concurrentes denominados *propagadores*— son completamente ortogonales al modelo de computación de Mozart, lo que garantiza una interacción transparente con las restricciones clásicas provistas por el lenguaje. La segunda dirección consta de un estudio de las implicaciones prácticas del esquema teórico de abstracción propuesto por Codognet et al., el cual plantea algunas claves que pueden simplificar los procesos de solución de CSPs débiles. Basado en estos resultados teóricos, un esquema iterativo de solución para Mozart es propuesto e implementado.

Para validar los componentes desarrollados se proponen casos de estudio tomados de problemas de la vida real. Se realizan comparaciones y análisis de los componentes entre sí y con respecto a otros enfoques. La aplicabilidad de los componentes con respecto a una categorización de CSPs es también estudiada. De esta forma, por los modelos e implementaciones propuestas, este trabajo contribuye al estado del arte en programación por restricciones dando una idea concreta de la aplicabilidad de las restricciones débiles en lenguajes de programación concurrente por restricciones.

Abstract

A wide variety of problems can be modeled as Constraint Satisfaction Problems (or CSPs). In areas such as planning, to describe a problem in terms of the mandatory conditions and the available resources turns out to be extremely convenient. However, there are situations that can not be faithfully represented by using this *classical* formulation of CSPs. For instance, those problems stating preferences or costs over the constraints, or those problems asserting contradictory constraints in such a way that it is impossible to find a suitable solution for the problem. In these cases, in order to obtain a solution, sometimes it is necessary to consider only a subset of the original constraints or to satisfy them in an incomplete manner. Several works have addressed this problem. The main theoretical proposals try to devise a *generic* framework that subsumes several kinds of CSPs. Other (rather practical) works provide ad-hoc solutions that albeit useful, are not applicable to other contexts.

This work focuses on one formal framework for handling the so-called *soft constraints*, or constraints that may be satisfied in a relaxed manner. More precisely, we study a framework for soft constraints that is based on a mathematical structure known as *semiring*. Our aim is to devise tools that enable Mozart, a concurrent constraint programming language, to effectively use soft constraints constructs in its programs. We believe the armonic integration of these tools within the concurrent model of Mozart will allow an straighforward inclusion of soft constraints statements in real-life constraint problems.

We follow a two-fold approach to conduct this study. Firstly, we propose a set of soft constraints for Mozart, based on a conservative modification to the original semiring-based model for soft constraints. Since these constraints are implemented as concurrent agents known as *propagators*, they are highly compatible with Mozart's computation model. Therefore, both soft and hard constraints can be transparently integrated in Mozart programs. Secondly, we study a theoretical proposal for abstracting soft constraints. In certain cases, this proposal provides hints that may help to speed up solving processes. We develop an iterative procedure that implements this proposal for Mozart and evaluate its performance.

The software components developed in this work are analyzed in real-life case studies. Moreover, comparisons and analysis of the components w.r.t. each other as well as different approaches are performed. The applicability of these components to the members of a category of CSPs is also studied. In this way, the models and tools proposed in this thesis contribute to the state of the art in constraint processing by giving a very concrete idea of the applicability of soft constraints techniques in concurrent constraint programming environments.

1 Introducción

Este trabajo de grado estudia el modelo de restricciones débiles basado en semianillos en el contexto de la programación concurrente por restricciones. En particular, se propone que los fundamentos de dicho modelo pueden ser transparentemente integrados con un lenguaje de programación *concurrente* por restricciones, de forma tal que los componentes resultantes de dicha integración puedan ser usados de forma efectiva para solucionar problemas de la vida real.

1.1. Motivación

Una gran variedad de problemas de la vida real (particularmente combinatorios) pueden ser modelados como *problemas de satisfacción de restricciones* (o CSPs). Este tipo de problemas puede entenderse como la descripción precisa de una serie de condiciones (denominadas *restricciones*) y de la información concerniente a los recursos disponibles para satisfacerlas. Este acercamiento *declarativo* a la solución de problemas resulta muy conveniente en una gran variedad de áreas, desde planeación y asignación de recursos hasta informática musical, pasando por computación gráfica y reconfiguración de sistemas. La *programación por restricciones* se ha constituido en el área que estudia los formalismos, técnicas y herramientas para la solución eficiente de CSPs.

Desde sus comienzos, la programación por restricciones ha asumido un criterio absoluto de satisfacción bajo el cual las restricciones en un CSP se cumplen o no, por lo que éstas se consideran *restricciones fuertes*. Este enfoque *clásico*, aunque apropiado para un buen número de casos, es inconveniente en dos situaciones particulares. Primero, criterios *débiles* como niveles de preferencia y costos sobre las restricciones, muy frecuentes en descripciones "humanas" de la realidad, son difíciles de acoplar en este estilo rígido de satisfacción. Segundo, una cantidad significativa de problemas de la vida real tienen restricciones que son contradictorias entre sí, por lo que encontrar una solución que las satisfaga a todas es imposible. En muchas ocasiones,

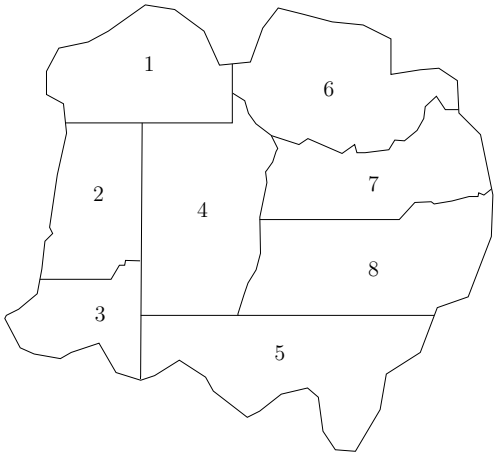

Figura 1.1: Un mapa divido en 8 regiones

estas restricciones contradictorias no son producto de un modelo defectuoso sino un reflejo de los requerimientos particulares del problema. La idea general detrás de estos problemas, denominados *sobre-restringidos*, es ilustrada en el siguiente ejemplo.

Ejemplo 1.1. *El* problema de coloreamiento de mapas *es uno de los CSPs más conocidos. Dado un mapa divido en regiones, el problema consiste en colorearlas utilizando un número limitado de colores, con la condición de que dos regiones adyacentes no pueden tener el mismo color. Considere la figura 1.1 y dos colores diferentes: blanco y negro. Por las características del mapa, no es posible cumplir con la restricción de coloreamiento. Por esta razón, y a pesar de que se trata de un problema relativamente pequeño, es imposible encontrar una solución que satisfaga todas las restricciones.*

En términos generales, resulta coherente pensar que las posibilidades de aplicación práctica de la programación por restricciones son enormes. Si se considera que hasta el momento la mayoría de aplicaciones y trabajos están basados en modelos rígidos de las situaciones reales, entonces la integración de criterios débiles en lenguajes y herramientas de programación por restricciones puede incrementar significativamente la aplicabilidad de esta tecnología.

Esta oportunidad ha sido identificada por la comunidad científica e interpretada en dos corrientes principales de investigación. La primera, que puede denominarse *práctica*, se ha interesado en la inclusión de criterios débiles a partir de las características prácticas asociadas con cada problema. Como consecuencia, diversos tipos de CSPs que representan diferentes formas de realizar tal inclusión han sido propuestos. Los CSPs *difusos* [DFP93], que integran los principios de la lógica difusa [KY95] en el contexto de la satisfacción de restricciones y los

CSP *probabilísticos* [FL93], que permiten razonar sobre la probabilidad de satisfacción de una restricción son sólo dos instancias específicas surgidas del acercamiento práctico.

Si bien el uso de estos tipos de CSPs puede resultar apropiado para ciertos casos particulares, es difícil pensar en alguno de los tipos de problemas surgidos del acercamiento práctico como una *técnica* de solución para CSPs débiles. En este sentido, la corriente *teórica* ha extendido el concepto original de CSP para hacerlo acorde con la inclusión de nuevos elementos (por lo general numéricos) que representen distintos criterios débiles. A diferencia de los trabajos en la corriente práctica, uno de los objetivos de los trabajos teóricos es ofrecer un *marco general* de problemas que sea lo suficientemente flexible para ajustarse a las características de diversos tipos de CSPs. La ventaja fundamental de este acercamiento es que permite constituir una base teórica sólida desde la cual dichos CSPs pueden derivar propiedades y resultados útiles.

La pregunta sobre cuál de las dos corrientes debería prevalecer al final es muy interesante. Por un lado, es claro que por su generalidad y propiedades los acercamientos teóricos son más confiables que las iniciativas particulares. En contra del acercamiento teórico debe considerarse que, como sucede en otras áreas de las ciencias de la computación, la brecha existente entre los resultados teóricos y las técnicas de programación de uso generalizado en la industria y la academia es muy amplia, lo que de alguna forma resta legitimidad a los resultados teóricos, por mas concluyentes y generales que estos sean.

El presente trabajo de grado pretende dar elementos de juicio para ponderar si el uso del acercamiento teórico para la solución de problemas de restricciones débiles resulta apropiado en situaciones de la vida real, dentro del contexto particular de la programación *concurrente* por restricciones. Más concretamente, se plantea que una de las propuestas teóricas para la idea de restricciones débiles —el modelo basado en *semianillos* [Bis04]— tiene las mismas posibilidades de aplicación que aquellos trabajos enmarcados en la corriente práctica. Esta tesis se pone a prueba por medio de dos acercamientos: en el primero se implementa una adaptación conservativa de las ideas principales del modelo, mientras que en el segundo un esquema de simplificación (conocido como *abstracción*) es llevado a la práctica. Ambas implementaciones están condicionadas por el hecho de que la relajación inducida por el modelo basado en semianillos se concentra en *debilitar* las condiciones que las restricciones imponen, en lugar de removerlas sistemáticamente. En consecuencia, este trabajo tiene los siguientes objetivos:

1. Desarrollar los componentes necesarios que permitan trabajar con una instancia del modelo de restricciones débiles basado en semianillos en Mozart-Oz.

2. Implementar un esquema de abstracción que resuelva un problema expresado como un semianillo utilizando el solucionador de dominios finitos de Mozart.

3. Contrastar el comportamiento de la abstracción en lo práctico con los beneficios expuestos en la teoría.

4. Comparar, en términos del tiempo de ejecución, los componentes del semianillo desarrollados para Mozart con el esquema de abstracción para diferentes tipos de problemas, realizando una categorización de éstos.

1.2. Contribuciones

Las contribuciones principales de este trabajo se condensan a continuación.

1. Una modificación teórica del modelo de restricciones débiles basado en semianillos que es susceptible a una implementación en cualquier lenguaje de programación por restricciones basado en técnicas de consistencia local.

2. Un conjunto de propagadores de restricciones débiles en Mozart que siguen este modelo modificado, y un conjunto de criterios de distribución específicos para problemas que incluyan restricciones débiles.

3. Un esquema iterativo de abstracción que usa restricciones clásicas (fuertes, provistas por Mozart) para solucionar problemas de satisfacción de restricciones difusos. Esto incluye un razonamiento formal sobre la relación entre propagadores clásicos relajados (base del esquema de abstracción) y los propagadores de restricciones débiles (mencionados en el punto anterior).

4. Conclusiones acerca de la aplicabilidad y comportamiento en la práctica de los componentes implementados. Estas conclusiones incluyen tanto análisis de desempeño (en términos de tiempo de ejecución) como una categorización de los problemas de satisfacción de restricciones de acuerdo a su posibilidad de ser atacados con restricciones débiles.

1.2.1. Publicaciones asociadas con este trabajo

Algunos de los resultados presentados en este trabajo de grado fueron publicadas previamente como contribuciones arbitradas (*peer-reviewed*) en eventos académicos internacionales.

- Alberto Delgado, Carlos A. Olarte, Jorge A. Pérez y Camilo Rueda. *Semiring-based Fuzzy Constraints in Concurrent Constraint Programming*. Memorias de la 31a Conferencia Latinoamericana de Informática (CLEI 2005), Cali (Colombia), Octubre de 2005. [DOPR05b].

  Una versión inicial de las modificaciones formales del capítulo 3 es descrita en este artículo.

- Alberto Delgado, Jorge A. Pérez y Camilo Rueda. *Implementing an Abstraction Framework for Soft Constraints*. Memorias del Sexto Simposio Internacional en Abstracción, Reformulación y Aproximación (SARA 2005), Airth (Escocia), Julio de 2005. [DPR05].

  El capítulo 5 está basado en el contenido de este artículo.

- Alberto Delgado, Carlos A. Olarte, Jorge A. Pérez y Camilo Rueda. *Implementing Semiring-Based Constraints Using Mozart*. Memorias de la Segunda Conferencia Internacional de Usuarios de Mozart/Oz (MOZ 2004), Charleroi (Bélgica), Octubre de 2004. [DOPR05a].

  Parte del contenido de los capítulos 4 y 6 está basado en los resultados de este artículo.

- Alberto Delgado, Carlos A. Olarte, Jorge A. Pérez y Camilo Rueda. *Implementing Semiring-Based Constraints Using Concurrent Constraint Programming*. Memorias del Sexto Workshop Internacional en Restricciones Débiles (SOFT 2004), llevado a cabo en el marco de la Décima Conferencia Internacional en Principios y Práctica de Programación por Restricciones (CP 2004), Toronto (Canadá), Septiembre de 2004. [DOPR04].

  Este artículo refleja una etapa inicial de los resultados presentados en el capítulo 4.

Adicionalmente, el módulo de semianillos descrito aquí fue desarrollado con base en las ideas de un módulo prototipo desarrollado conjuntamente por Carlos Olarte y los autores, y reportado en [DOPR04] y [DOPR05a].

1.3. Estructura del documento

Una breve descripción de los fundamentos de la programación por restricciones es presentada en el capítulo segundo (Marco Teórico). Dichos conceptos se trasladan al contexto de la programación *concurrente* por restricciones, el área de interés en este trabajo. Dicho capítulo presenta también el formalismo de restricciones débiles basado en semianillos. Esta presentación incluye la descripción del esquema de abstracción existente para este formalismo.

Los conceptos formales son transformados conservativamente en el tercer capítulo (De la Teoría a la Práctica). Esta transformación adapta algunas nociones del modelo de semianillos con el fin de hacerlo más cercano al modelo de propagación subyacente a la mayoría de lenguajes de programación por restricciones. Con dicha adaptación en mente, un algoritmo iterativo que aplica los conceptos relacionados con el esquema de abstracción es propuesto. Ambos resultados (la adaptación del modelo y el algoritmo que soporta el esquema de abstracción) son sustentados formalmente.

Los capítulos cuarto y quinto describen la implementación de estos resultados. Específicamente, el capítulo cuarto (Un Solucionador para CSPs Difusos) se ocupa de presentar las principales características de un módulo de restricciones débiles para Mozart, que implementa un solucionador para problemas de satisfacción de restricciones difusas. Este módulo incluye elementos de propagación y criterios de distribución. El capítulo presenta también un análisis práctico del módulo. Por su parte, el capítulo quinto (Un Esquema de Abstracción para CSPs Difusos), describe los detalles más relevantes de la implementación del esquema iterativo de abstracción para Mozart. Sus principales características, así como una evaluación de su rendimiento son descritos.

El capítulo sexto (Análisis y Comparaciones) presenta un análisis de las dos técnicas diseñadas. Este análisis clasifica los tipos de problemas que pueden ser atacados usando restricciones débiles y enumera algunas de las condiciones y consideraciones que deben tenerse en cuenta para este fin. Comparaciones que involucran los elementos implementados son también incluidas. Finalmente, el capítulo séptimo presenta una serie de consideraciones finales resultantes de este trabajo. Una completa revisión de trabajo relacionado es incluida allí.

2 Marco Teórico

Este capítulo provee el marco conceptual que soporta el desarrollo de la tesis. Está dividido en dos partes: la primera presenta las nociones fundamentales del modelo de programación por restricciones. Dichas nociones son estudiadas luego desde la perspectiva de la programación *concurrente* por restricciones, modelo teórico de interés en este trabajo. La segunda parte se ocupa de recopilar los fundamentos y propiedades del modelo de restricciones débiles basado en semianillos. Un esquema de *abstracción* propuesto para este modelo es también presentado. Bajo ciertas condiciones, este esquema puede simplificar la solución de problemas de satisfacción de restricciones débiles.

2.1. Programación por Restricciones

La programación por restricciones es una de las ramas de la programación de más importante desarrollo en la última década. Este paradigma de programación tiene un fuerte componente teórico, apoyado en la lógica y la inteligencia artificial y poco a poco se ha ido imponiendo como una alternativa para desarrollar soluciones comerciales a problemas clásicos de optimización y de planeación (especialmente combinatorios), que involucran diferentes tipos de restricciones. Por tal razón, la programación por restricciones ha sido identificada por la ACM (Association for Computing Machinery) como una de las direcciones estratégicas de investigación en computación [MS98]. Su interés concreto es dar solución a un conjunto de problemas bien definido, abundantes en la vida práctica, los *problemas de satisfacción de restricciones* (o CSPs por sus siglas en inglés).

En general, un CSP incluye dos componentes principales: *variables* con dominios asociados y *restricciones*. Mientras que las primeras son objetos que pueden tomar diversos valores, las segundas son reglas que imponen una limitación sobre las valores que una variable, o una combinación de variables, pueden tener asignadas [Dec03]. El siguiente ejemplo, tomado de [Apt03], ilustra esta idea intuitiva.

Ejemplo 2.1 ($SEND + MORE = MONEY$). *Este es el ejemplo clásico por excelencia de la programación por restricciones. En este problema, el objetivo es encontrar dígitos distintos para las letras D, E, M, N, O, R, S, Y de modo que la suma*

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array}$$

se satisfaga. De esta forma, las variables del problema son las letras presentes en la suma. El dominio de las variables son los enteros de 0 a 9, con excepción de las letras S y M, que por ser los primeros dígitos, deben ser diferentes de cero y tienen como dominio los enteros de 1 a 9.

Existen diversas formulaciones para este problema. Una de ellas establece la restricción de igualdad

$$\begin{aligned} & 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\ & 1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\ = 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y, & \end{aligned}$$

combinada con 28 restricciones de desigualdad $i \neq j$ en donde i, j representan a los elementos del conjunto $\{D, E, M, N, O, R, S, Y\}$.

A continuación presentamos —de un modo más formal— las nociones principales del modelo de programación por restricciones, con base en [Bar99].

Definición 2.1 (Problema de satisfacción de restricciones (CSP)). *Un CSP puede definirse de la siguiente manera :*

- *Un conjunto de* variables $\mathcal{X} = \{X_1, X_2, ..., X_n\}$,
- *un conjunto finito D_i de posibles valores (dominio) para cada variable $X_i \in \mathcal{X}$,*
- *un conjunto C de* restricciones*, que condicionan los valores que las variables pueden tomar simultáneamente.*

Los dominios de las variables pueden ser cualquier conjunto, finito o infinito. En la práctica, y para asegurar decidibilidad, se utilizan restricciones de dominios finitos, que sirven para

modelar espacios de búsquedas y se expresan como intervalos. Por ejemplo, la expresión $X \in [l, u]$ establece que los valores posibles para la variable entera X son todos los números enteros entre l y u. Las restricciones también pueden ser de diversos tipos, por ejemplo, restricciones lógicas para expresar condiciones sobre las variables, restricciones globales que se imponen sobre todo el conjunto de variables, entre otras.

Una *solución* para un CSP es una asignación de valores para toda variable X_i tal que todas las restricciones en C se satisfacen simultáneamente. Dependiendo de la situación puede desearse *una sola solución* sin importar cuál, *todas las soluciones* posibles o una *solución óptima*, dada alguna función objetivo definida en términos de un conjunto de las variables del problema.

Las soluciones para un CSP pueden encontrarse buscando sistemáticamente en el espacio de posibles asignaciones de valores a variables. Los métodos de búsqueda pueden dividirse en dos clases, aquellos que recorren el espacio de soluciones parciales (o asignaciones parciales de valores) y aquellos que exploran el espacio de asignaciones completas de valores de forma estocástica. Para los efectos de este trabajo, nos interesa describir los primeros.

2.1.1. Búsqueda Sistemática de Soluciones para un CSP

Desde un punto de vista teórico, resolver un CSP resulta trivial si se explora sistemáticamente el espacio de soluciones. Desde un punto de vista práctico, en el que la eficiencia cuenta, esto no resulta tan cierto. Aun cuando los métodos sistemáticos pueden parecer simples e ineficientes, son importantes porque conforman la base de algoritmos más sofisticados y eficientes.

El algoritmo de satisfacción de restricciones básico, que recorre el espacio de asignaciones completas, se conoce como generar y probar (*generate and test*, GT). La idea del algoritmo es simple: primero se *genera* aleatoriamente una asignación de variables completa, y seguidamente, se *prueba* si dicha asignación satisface todas las restricciones. En caso afirmativo, se ha encontrado una solución. De otro modo, otra asignación es generada.

El algoritmo generar y probar es un algoritmo genérico pobre que se usa como último recurso, en caso de que otros métodos fallen. Resulta ineficiente porque el generador de asignaciones no posee ningún tipo de información y descubre tarde las inconsistencias. Esto sugiere dos mejoras para el algoritmo:

- Un generador de valuaciones inteligente (informado), que genere asignaciones completas de modo que los conflictos con la fase de prueba sean minimizados. Esta es la idea detrás

de algoritmos de búsqueda local como *hill-climbing* [MF02].

- La combinación del generador y la fase de prueba: la validez de la restricción es comprobada tan pronto como sus respectivos valores son asignados. Esta idea es la base del algoritmo *backtracking*.

El método del algoritmo backtracking consiste en extender incrementalmente una solución parcial (que especifica valores válidos para algunas de las variables) a una solución completa, escogiendo valores consistentes (con respecto a la solución parcial) para las variables sin instanciar. Más específicamente, el algoritmo asigna secuencialmente valores a las variables y tan pronto como todas las variables referentes a una restricción tengan un valor, la validez de las restricciones es verificada. Si una solución parcial no satisface alguna de las restricciones, se ejecuta un *retroceso* hacia la variable asignada más reciente que aún tenga alternativas disponibles. Claramente, en cualquier momento en que una asignación parcial de variables viola una restricción, el algoritmo backtracking está en la capacidad de eliminar un subespacio del producto cartesiano de todos los dominios de las variables. Consecuentemente, backtracking puede considerarse estrictamente mejor que el algoritmo de generar y probar.

De la descripción anterior se pueden identificar dos desventajas principales del algoritmo backtracking estándar: falla en repetidas ocasiones por la misma razón (debido a que no tiene un mecanismo de registro de conflictos), lo que hace que el algoritmo repita trabajo, y no provee una detección temprana de conflictos, desperdiciando recursos.

2.1.2. Técnicas de consistencia

Para detectar de manera anticipada las posibles inconsistencias, se han desarrollado las denominadas *técnicas de consistencia* [Mac77]. La idea principal de estas técnicas es remover valores inconsistentes de los dominios de las variables hasta que se encuentre una solución. Se trata de un mecanismo determinístico, en contraste a la búsqueda no determinista. La mayoría de técnicas de consistencia son *incompletas*, es decir, por si solas no encuentran una solución, por lo que raramente son usadas de manera única para resolver un CSP.

Las características principales de las técnicas básicas de consistencia se derivan de las nociones de grafo. Generalmente un CSP se representa como un *grafo de restricciones*, donde los nodos corresponden a las variables y los arcos que las unen están descritos por restricciones. Esto requiere que el CSP esté escrito de una manera especial conocida como CSP binario (es decir, un CSP que contiene solamente restricciones unarias y binarias). Es posible mostrar

que un CSP arbitrario tiene un equivalente binario [RPD90], aunque en el proceso se añaden demasiadas variables.

La técnica de consistencia más simple se conoce como *nodo consistencia*, y consiste en remover valores de los dominios de las variables que son inconsistentes con restricciones unarias sobre la variable respectiva.

La técnica de consistencia más utilizada se denomina *arco consistencia*. Esta técnica remueve valores de los dominios de las variables que son inconsistentes con restricciones binarias. En particular, el arco (X_i, X_j) es arco consistente si y sólo si por cada valor x en el dominio actual de X_i que satisface las restricciones sobre X_i existe algún valor y en el dominio de V_j tal que las asignaciones $V_i = x$ y $V_j = y$ son permitidas por la restricción binaria entre X_i y X_j.

Una técnica que remueve muchos más valores es *camino consistencia*. Para cada par de valores de dos variables X y Y que satisfacen la restricción binaria respectiva, camino consistencia requiere la existencia de un valor para cada variable a lo largo de algún camino entre X y Y, de modo que todas las restricciones en dicho camino sean satisfechas. Se demostró que un CSP es camino consistente si y sólo si todas las rutas de longitud 2 son camino consistentes [Mon74]. Por lo tanto, los algoritmos de camino consistencia pueden trabajar con triplas de variables (caminos de longitud 2). Existen diversos algoritmos de camino consistencia, pero se caracterizan por consumir mucho espacio en memoria.

Todas las técnicas de consistencia mencionadas son generalizadas por la idea de *k-consistencia*. Un grafo de restricciones es k-consistente si por cada sistema de valores para $k-1$ variables que satisfacen todas las restricciones entre esas variables, existe un valor para una k-ésima variable arbitraria tal que las restricciones entre las k variables se satisfacen.

Como se dijo antes, las técnicas de consistencia son incompletas, pues puede que después de aplicar los algoritmos algunos valores inconsistentes permanezcan en los dominios o no se llegue a una única solución. Para estos casos, existen técnicas que *asignan* un valor a una variable especifica, con el objetivo de lograr que por medio de las relaciones existentes entre la variable elegida y las demás variables del problema, los dominios de todas las variables del problema se reduzcan. Estas técnicas de *enumeración* son por lo tanto complementarias con las técnicas de consistencia local, y deben alternarse en la búsqueda de una solución.

La programación por restricciones comprende a su vez distintas áreas. Algunos esquemas de solución de los problemas de satisfacción de restricciones se han convertido en sub-áreas o

ramas de la programación por restricciones, construyendo un marco teórico propio alrededor de un concepto o paradigma especifico. Por ejemplo, la *programación lógica por restricciones* aprovecha el esquema declarativo de los lenguajes de programación lógica (como Prolog) y su sistema de inferencia para solucionar restricciones expresadas como predicados de primer orden. A continuación se describen los fundamentos teóricos de otra rama de la programación por restricciones, la *programación concurrente por restricciones*, área en la que el presente trabajo está inscrito.

2.1.3. Programación por Restricciones en Mozart

Mozart es un lenguaje de programación por restricciones basado en el lenguaje Oz [Smo95]. Mozart provee diversas funcionalidades, incluyendo soporte para programación distribuida, aplicaciones multi-agente, satisfacción de restricciones, así como herramientas de apoyo al programador. Está concebido sobre el paradigma de programación *concurrente* por restricciones (CCP, por sus siglas en inglés) [SRP91], propuesto por Vijay Saraswat a comienzos de la década pasada. En este modelo de computación se privilegia la noción de restricción como elemento de *información parcial*, con base en dos componentes principales: un *almacén de restricciones* que acumula toda la información del sistema, y un conjunto de *agentes o procesos* que pueden adicionar o consultar información sobre dicho almacén. Los agentes actúan concurrentemente, y su sincronización depende de la información disponible en el almacén.

Mozart considera dos tipos de restricciones: básicas y no básicas. Una *restricción básica* es una fórmula lógica interpretada en alguna estructura de primer orden. Por ejemplo, en el contexto de las variables de dominio finito, $x \in D$ (en donde x es una variable y D es un dominio) es una restricción básica. Por otra parte, las *restricciones no básicas* son combinaciones de restricciones básicas. Con el fin de procurar que las operaciones sobre las restricciones sean eficientes, las restricciones no básicas no se incluyen en el almacén de restricciones, sino que son impuestas por *propagadores*. Un propagador es un agente computacional que encapsula una función de filtrado de valores que traduce una restricción no básica en restricciones básicas (incluibles en el almacén de restricciones). Un propagador para una restricción c deja de existir tan pronto como c se deduzca del almacén de restricciones, o si la conjunción del almacén de restricciones actual y c es insatisfactible. En este ultimo caso, se dice que el propagador de c no se deduce, al deducirse $\neg c$ del almacén actual [MÖ01]. Por lo general hay muchos propagadores impuestos sobre un conjunto de variables, lo cual propicia que diferentes propagadores tengan información común sobre las variables usando el almacén como medio

compartido. Esto ocasiona que los propagadores se activen unos a otros por la escritura de restricciones básicas al almacén de restricciones. Este proceso continúa hasta que se alcance un punto fijo en la propagación; es decir, hasta cuando no se puedan escribir restricciones básicas adicionales a las existentes. El orden en el cual los propagadores adicionan la información al almacén de restricciones es irrelevante en el resultado final.

Los procesos de solución en Mozart tienen lugar en *espacios computacionales*. Un espacio computacional está compuesto de un conjunto de propagadores conectados con un almacén de restricciones. Se dice que un espacio es *estable* si no es posible adelantar propagación adicional. Un espacio estable S se denomina *fallido* si algún propagador deduce la negación de alguna restricción, o *resuelto* si S no contiene propagadores. Adicionalmente, una asignación de variables se denomina *solución* para un espacio si dicha asignación satisface las restricciones en el almacén y todas las restricciones impuestas por medio de propagadores.

Como se mencionaba antes, los procesos de propagación no constituyen un método completo de solución. Eso significa que un espacio computacional puede tener una o varias soluciones y que la propagación no las encuentre. También puede suceder que dicho espacio no tenga solución y que la propagación no derive en un propagador fallido. En estos casos, el espacio debe ser *distribuído*. Dado un espacio de computación (ni fallido ni resuelto), se elige una restricción c, y se crean dos nuevos espacios a solucionar: $S \wedge c$ y $S \wedge \neg c$. Es crucial elegir c de forma tal que los espacios nuevos generen nuevos procesos de propagación. Las restricciones $S \wedge c$ y $S \wedge \neg c$, denominadas *alternativas*, son por lo general restricciones de igualdad, i.e., $c = (x = n)$ y $\neg c = (x \neq n)$. De esta forma, se obtiene un *árbol de búsqueda*, en donde cada nodo corresponde a un espacio y cada hoja del árbol corresponde a un espacio de computación resuelto o fallido. Por la relación existente entre las variables del problema y las alternativas, se asumen árboles de búsqueda finitos.

Un *distribuidor* es un agente que implementa una estrategia de distribución. De forma similar a las técnicas de enumeración mencionadas antes, esta estrategia está definida sobre una secuencia de variables $x_1, \ldots x_n$. Cuando un paso de distribución es necesario, la estrategia selecciona una variable en la secuencia por determinar y distribuye sobre esta variable, i.e., impone una restricción sobre la variable seleccionada. Existen diversas posibilidades para distribuir una variable. Por ejemplo, una estrategia de distribución *ingenua (naive)* selecciona la variable no determinada ubicada más a la izquierda dentro de la secuencia, mientras que una estrategia de distribución *first fail* selecciona la variable no determinada —ubicada más a la izquierda dentro de la secuencia— que tenga el dominio más pequeño.

2.2. Restricciones Débiles basadas en Semianillos

El modelo de restricciones débiles basado en semianillos asocia un semianillo a la definición estándar de problema de restricciones, de tal modo que diferentes escogencias de los componentes del semianillo pueden representar distintos tipos de problemas de satisfacción de restricciones. El semianillo provee un conjunto de elementos que se asocian con las restricciones (denominadas *valuaciones*) y las operaciones permitidas sobre dichos elementos. A continuación se presenta un compendio de este modelo; mayores detalles pueden obtenerse en [Bis04].

Definición 2.2. *Un* ***semianillo de restricciones*** *es una tupla* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *tal que:*

- A *es un conjunto y* $\mathbf{0}, \mathbf{1} \in A$
- $+$ *—la* ***operación aditiva****— cumple con las propiedades clausurativa, conmutativa y asociativa. Adicionalmente, esta operación tiene como elemento unitario a* $\mathbf{0}$ *(esto es, para todo* $a \in A$, $a + \mathbf{0} = a$*),* 1 *es su elemento absorbente (es decir, para todo* $a \in A$, $\mathbf{1} + a = \mathbf{1}$*) y es idempotente.*
- $\times$*, denominada la* ***operación multiplicativa****, cumple con las propiedades conmutativa, clausurativa y asociativa de modo que* 1 *es su elemento unitario y* $\mathbf{0}$ *es su elemento absorbente.*
- $\times$ *distribuye sobre* $+$ *(es decir,* $a \times (b + c) = (a \times b) + (a \times c)$*)*

La idempotencia de la operación $+$ es necesaria para definir un orden parcial $\leq_S$ sobre el conjunto A, que permita comparar diferentes elementos del semianillo. Este orden parcial está definido como $a \leq_S b$ si y sólo si $a + b = b$. Intuitivamente, $a \leq_S b$ quiere decir que b "es mejor"que a. Es posible extender esta idea de comparación para abordar el concepto de "mejor solución", si se considera que entre a y b, la operación aditiva *escoge* b. La conmutatividad de $\times$ es necesaria para combinar consistentemente varias restricciones, garantizando el mismo resultado sin importar el orden en que se impongan las restricciones.

Adicionalmente, si $\mathbf{1}$ es el elemento absorbente de la operación $+$, entonces se tiene que $a \leq_S \mathbf{1}$ para todo a. Por tanto, puede considerarse que $\mathbf{1}$ es el *máximo* (mejor) elemento del ordenamiento parcial. Esto implica que la operación $\times$ es *intensiva*, es decir, se cumple que $a \times b \leq_S a$. En términos simples, esto significa que la combinación de restricciones conduce a *peores resultados* (en términos del orden parcial $\leq_S$). El hecho de que $\mathbf{0}$ sea el elemento

unitario de la operación aditiva implica que es el *mínimo* (peor) elemento del ordenamiento. Por lo tanto, para cualquier $a \in A$, $\mathbf{0} \leq_S a \leq_S \mathbf{1}$.

2.2.1. Sistemas y Problemas de Restricciones

Un sistema de restricciones define el semianillo que debe utilizarse, el conjunto de variables V y su dominio D. Una restricción sobre un sistema de restricciones dado especifica las variables involucradas y los valores asociados con ellas. Más precisamente, para cada tupla de valores en el producto cartesiano de las variables involucradas, se define un elemento correspondiente del conjunto A. Este elemento puede ser interpretado como el peso o costo de la tupla, o como algún otro indicador cuantitativo. De esta forma, un problema de restricciones puede definirse como un conjunto de restricciones sobre un sistema de restricciones dado, sumado a un conjunto selecto de *variables de interés* de las que se desea conocer las asignaciones compatibles con todas las restricciones.

Definición 2.3 (*Sistema de Restricciones, Restricción, Problema*). *Un* sistema de restricciones *es una tupla $CS = \langle S, D, V \rangle$, donde S es un semianillo de restricciones, D es un conjunto finito y V es un conjunto ordenado de variables. Dado un sistema de restricciones $CS = \langle S, D, V \rangle$ (con $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$), una* restricción *sobre CS es una pareja $\langle def, \mathsf{con} \rangle$, donde con —denominado el tipo de la restricción— es subconjunto de V y def es una función $f : D^{|\mathsf{con}|} \to A$ (llamado el valor de la restricción) [Bis04].*

Adicionalmente, un problema de restricciones (o SCSP) *P sobre CS es una pareja $P = \langle C, \mathsf{con} \rangle$ donde C es un conjunto de restricciones sobre CS y $\mathsf{con} \subseteq V$.*

En un SCSP, los valores asociados con las tuplas que representan cada restricción son usados para calcular los valores correspondientes a las tuplas de las variables en con, de acuerdo con las operaciones del semianillo: la operación multiplicativa permite combinar los valores de las tuplas para obtener el valor de una tupla para todas las variables mientras que la operación aditiva permite obtener el valor de las tuplas de las variables de interés. Estas dos operaciones se pueden definir de un modo más preciso como las operaciones de *combinación* ($\otimes$) y *proyección* ($\Downarrow$) sobre restricciones.

Definición 2.4 (*Combinación y proyección*). *Considere cualquier tupla de valores t y dos conjuntos de variables I y I', donde $I' \subseteq I$. Para las variables en el conjunto I, $t \downarrow_{I'}^{I}$, denota la proyección de la tupla t sobre las variables del conjunto I'. Sean $c_1 = \langle def_1, \mathsf{con}_1 \rangle$*

y $c_2 = \langle def_2, \mathsf{con}_2 \rangle$ *dos restricciones sobre* CS*. Su* combinación $c_1 \otimes c_2$ *es la restricción* $c' = \langle def', \mathsf{con}' \rangle$*, donde* $\mathsf{con}' = \mathsf{con}_1 \cup \mathsf{con}_2$ *y* $def'(t) = def_1(t \downarrow_{\mathsf{con}_1}^{\mathsf{con}'}) \times def_2(t \downarrow_{\mathsf{con}_2}^{\mathsf{con}'})$*.*

Además, dada la restricción $c = \langle def, \mathsf{con} \rangle$ *y un subconjunto* w *de* con*, su* proyección *sobre* w *(escrita* $c \Downarrow_w$*) es la restricción* $\langle def^*, \mathsf{con}^* \rangle$*, donde* $\mathsf{con}^* = w$ *y* $def^*(t^*) = \sum_{\{t | t \downarrow_w^{\mathsf{con}} = t^*\}} def(t)$*.*

Con estas operaciones, es posible definir la noción de solución de un SCSP:

Definición 2.5 (*Solución (óptima) de un problema de restricciones*). *Dado un problema de restricciones* $P = \langle C, \mathsf{con} \rangle$ *sobre un sistema de restricciones* CS*, su* solución *es una restricción definida como* $Sol(P) = (\bigotimes C) \Downarrow_{\mathsf{con}}$ *donde* $\bigotimes C$ *es la extensión de la operación de combinación a un conjunto de restricciones* C*. Adicionalmente una* solución óptima *para* P *es una pareja* $\langle t, v \rangle$ *tal que* $def(t) = v$ *y que no existe un* t *tal que* $v \leq def(t)$*. El conjunto de soluciones óptimas de* P *se denota por* $Opt(P)$*.*

En otras palabras, la solución de un SCSP es la restricción inducida por el *problema completo* sobre las variables en con. Tal restricción provee, para cada tupla de valores del producto cartesiano de las variables en con, un valor asociado del conjunto A. Algunas veces resulta suficiente conocer únicamente el mejor valor asociado con dichas tuplas. Este valor, denominado el *mejor nivel de consistencia* del problema, puede obtenerse por la combinación de todas las restricciones y su posterior proyección sobre un conjunto vacío de variables:

Definición 2.6 (*Mejor nivel de consistencia*). *Dado un SCSP* $P = \langle C, \mathsf{con} \rangle$*, su mejor nivel de consistencia se define como la restricción* $blevel(P) = (\bigotimes C) \Downarrow_{\emptyset}$*. Se dice que* P *es consistente si* $\mathbf{0} <_S blevel(P)$ *y que es* α*-consistente si* $blevel(P) = \alpha$*.*

Informalmente, el mejor nivel de consistencia da una idea general de qué tanto pueden satisfacerse las restricciones de un problema dado. Es un valor independiente de las variables de interés.

Usando el ordenamiento $\leq_S$ sobre el semianillo, es posible definir un ordenamiento sobre las restricciones que tienen el mismo tipo.

Definición 2.7 (*Ordenamiento sobre restricciones*). *Sean* c_1, c_2 *dos restricciones sobre* CS *tal que* $\mathsf{con}_1 = \mathsf{con}_2$ *y* $|\mathsf{con}_1| = k$*. El orden* $c_1 \sqsubseteq_S c_2$ *existe si y sólo si para todas las* k*-tuplas* t *de valores de* D*,* $def_1(t) \leq_S def_2(t)$*. Nótese que si* $c_1 \sqsubseteq_S c_2$ *y* $c_2 \sqsubseteq_S c_1$*, entonces* $c_1 = c_2$*.*

A partir de este ordenamiento, y aprovechando el hecho de que una solución es una restricción, es posible definir un ordenamiento sobre problemas de restricciones.

Definición 2.8 (*Ordenamiento sobre problemas*). *Suponga dos SCSPs* $P_1 = \langle C_1, \mathsf{con}\rangle$ *y* $P_2 = \langle C_2, \mathsf{con}\rangle$ *sobre CS. Entonces,* $P_1 \sqsubseteq_P P_2$ *si* $Sol(P_1) \sqsubseteq_S Sol(P_2)$*. En el caso en que* $P_1 \sqsubseteq_P P_2$ *y* $P_2 \sqsubseteq_P P_1$*, ambos problemas tienen la misma solución. Se dice entonces que* P_1 *y* P_2 *son equivalentes, denotado con* $P_1 \equiv P_2$*.*

Un aspecto interesante es que el ordenamiento $\sqsubseteq_P$ puede ser usado para ordenar conjuntos de restricciones, pues un conjunto de restricciones es un problema en donde con contiene todas las variables. Por otra parte, el formalismo de restricciones débiles basado en semianillos es *monótono*. Esto significa que añadiendo nuevas restricciones, la solución del nuevo problema precede (en términos del orden $\sqsubseteq_S$) a la solución anterior. Este resultado se formaliza en el siguiente teorema.

Teorema 2.1. ***(Monotonicidad de un SCSP)*** *Para dos SCSPs* $P_1 = \langle C_1, \mathsf{con}\rangle$ *y* $P_2 = \langle C_1 \cup C_2, \mathsf{con}\rangle$ *sobre CS, se tiene que* $P_1 \sqsubseteq_P P_2$*, con* $blevel(P_2) \leq_S blevel(P_1)$*.*

2.2.2. Instancias del modelo

El modelo general de restricciones débiles basadas en semianillos puede instanciarse para representar diferentes tipos de problemas:

Semianillo clásico, $S_{CSP} = \langle\{0,1\}, \vee, \wedge, 0, 1\rangle$. Este semianillo modela los problemas de restricciones convencionales en donde una restricción se cumple o no se cumple. Las tuplas permitidas en el problema estarán asociadas con 1, mientras que las rechazadas estarán asociadas con 0.

Semianillo probabilístico, $S_{prob} = \langle\{x|\ x \in [0,1]\}, max, \times, 0, 1\rangle$. Este semianillo modela CSP probabilísticos, en donde se razona sobre la probabilidad que tiene una restricción de ser parte del problema dado. Al decir que una restricción c tiene probabilidad p significa que la situación que c representa tiene una probabilidad p de ocurrir en el problema real. El propósito es obtener aquellas instanciaciones de variables con la máxima probabilidad.

Semianillo de optimización (o semianillo *weighted*), $S_{WCSP} = \langle\mathcal{R}^+, min, +, +\infty, 0\rangle$. Este semianillo modela aquellos CSP en donde cada tupla tiene un costo asociado. Se

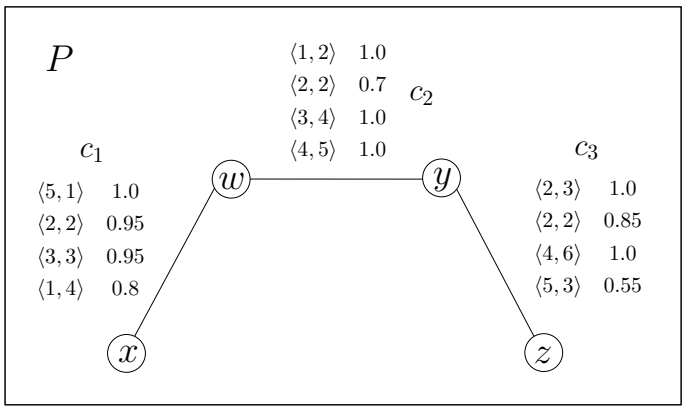

Figura 2.1: Un SCSP difuso

genera entonces un problema de optimización en donde el objetivo es minimizar el costo total de las soluciones. Dicho costo total se define como la sumatoria de los costos de todas las restricciones.

Semianillo difuso, $S_{FCSP} = \langle \{x|\ x \in [0,1]\}, max, min, 0, 1\rangle$. Este semianillo representa CSPs difusos, un tipo de problemas que extienden la noción clásica de problemas de restricciones asociando cada tupla de valores con un nivel de preferencia. Dicho nivel se encuentra siempre entre 0 y 1, en donde 1 representa el mejor valor (la tupla es aceptada) y 0 el peor (la tupla es rechazada completamente).

Ejemplo 2.2. *La figura 2.1 muestra P, un SCSP difuso. Además del semianillo* S_{FCSP}*, el sistema de restricciones está definido por* $V = \{x, y, w, z\}$ *y por D, que establece los dominios de las variables:* $dom(x) = \{1, 2, 3, 5\}$*,* $dom(y) = \{2, 4, 5\}$*,* $dom(w) = \{1, 2, 3, 4\}$ *y* $dom(z) = \{2, 3, 6\}$*. Las restricciones en P están dadas por* $C = \{c_1, c_2, c_3\}$ *y están definidas por extensión. Por ejemplo,* $c_2 = \langle def_2, \mathsf{con}_2\rangle$ *donde* $\mathsf{con}_2 = \{w, y\}$ *y*

$$def_2 = \{(\langle 1, 2\rangle, 1.0), (\langle 2, 2\rangle, 0.7), (\langle 3, 4\rangle, 1.0), (\langle 4, 5\rangle, 1.0)\}.$$

Una de las soluciones para P es $\langle 2, 2, 2, 2\rangle$ *con valuación 0.7. Tal valuación es el resultado de la minimización de las valuaciones de las tres subtuplas asociadas, i.e.,* $min(0.95, 0.7, 0.85)$.

2.2.3. Un Esquema de Abstracción para SCSPs

A continuación se presentan los resultados principales de una propuesta teórica reciente que tiene como objeto facilitar los procesos de solución para los problemas de satisfacción de

restricciones débiles por medio de su *abstracción*. Mayores detalles pueden encontrarse en [BCR02, BRP03].

2.2.3.1. Conceptos de Interpretación Abstracta

Se conoce como *interpretación abstracta (abstract interpretation)* [Cou96] al conjunto de teorías que han sido desarrolladas para razonar sobre la relación entre dos semánticas diferentes, una *concreta* y otra *abstracta*. Esta idea, originalmente propuesta para el análisis de programas, pretende obtener una semántica aproximada (la abstracta) a partir de una exacta (la semántica concreta). La semántica aproximada debe preservar la estructura y algunas de las propiedades presentes en la semántica concreta. Esta idea puede formalizarse por medio de un par de funciones, la *función de abstracción* (denotada por α) y la *función de concretización* (denotada por γ). Estas dos funciones conforman una *conexión de Galois*.

Sean $(\mathcal{C}, \sqsubseteq)$ el dominio de la semántica concreta y $(\mathcal{A}, \leq)$ su contraparte para la semántica abstracta. La idea de aproximación discutida arriba se hace explicita con la noción de *precisión* inducida por las relaciones de orden parcial. Más específicamente, si α es la función que asocia un elemento en $(\mathcal{A}, \leq)$ para cualquier elemento en $(\mathcal{C}, \sqsubseteq)$, se puede asegurar lo siguiente: si $\alpha(x) \leq y$, entonces y es una aproximación abstracta correcta de x, aunque *menos precisa* que $\alpha(x)$. Similarmente para $x \sqsubseteq \gamma(y)$: en este caso, x provee información *más precisa* que $\gamma(y)$. Todo lo anterior da origen a la siguiente definición formal.

Definición 2.9 (*Conexión e Inserción de Galois*). *Sean $(\mathcal{C}, \sqsubseteq)$ y $(\mathcal{A}, \leq)$ dos conjuntos parcialmente ordenados. Una* conexión de Galois $\langle \alpha, \gamma \rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ *es un par de funciones $\alpha : \mathcal{C} \to \mathcal{A}$ y $\gamma : \mathcal{A} \to \mathcal{C}$ tales que: 1. α y γ son monótonas, 2. para cada $x \in \mathcal{C}$, $x \sqsubseteq \gamma(\alpha(x))$ y 3. para cada $y \in \mathcal{A}$, $\alpha(\gamma(y)) \leq y$. Adicionalmente, una* inserción de Galois *(de $\mathcal{A}$ en $\mathcal{C}$) $\langle \alpha, \gamma \rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ es una conexión de Galois donde $\gamma \cdot \alpha = Id_{\mathcal{A}}$.*

La figura 2.2 ilustra una inserción de Galois para un dominio concreto $\langle [0,1], \leq \rangle$ y un dominio abstracto $\langle \{0,1\}, \leq \rangle$. Dado un número real $0 < \theta < 1$, α asocia todos los números reales en el intervalo $[0, \theta]$ con 0, y todos los demás reales con 1. La función γ asocia 0 con θ y 1 con 1.

Por lo general, tanto el dominio concreto como el dominio abstracto tienen operadores asociados con cada una de las semánticas. Por esto, resulta útil definir de forma precisa la relación existente entre los operadores sobre el dominio abstracto con los operadores del dominio concreto. Esta relación se define a continuación.

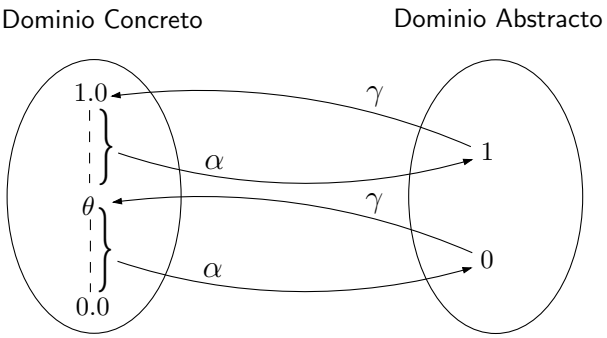

Figura 2.2: Una Inserción de Galois

Definición 2.10 (*Correctitud Local*). *Sean $f : \mathcal{C}^n \to \mathcal{C}$ un operador sobre el dominio concreto y $\tilde{f}$ su contraparte abstracta. Se dice que $\tilde{f}$ es localmente correcto con respecto a f si $\exists x_1, \ldots, x_n \in \mathcal{C}, f(x_1, \ldots, x_n) \sqsubseteq \gamma(\tilde{f}(\alpha(x_1), \ldots, \alpha(x_n)))$.*

Intuitivamente, esta propiedad significa que $\tilde{f}$ aproxima de manera segura a f, aunque con un menor nivel de precisión.

2.2.3.2. Abstracción para SCSPs

A partir de las definiciones anteriores, es posible definir un esquema de abstracción para problemas de satisfacción de restricciones débiles. La idea consiste en pasar, por medio de la abstracción, de un SCSP P sobre un semianillo S a otro SCSP $\tilde{P}$ sobre el semianillo de restricciones $\tilde{S}$, en donde $\leq_S$ y $\leq_{\tilde{S}}$ (los ordenes parciales asociados con S y $\tilde{S}$, respectivamente) están relacionados entre sí por una inserción de Galois.

Definición 2.11 (*Abstracción para un SCSP*). *Suponga un SCSP concreto $P = \langle C, \mathsf{con}\rangle$ sobre el semianillo de restricciones S, en donde*

- $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *y*
- $C = \{c_0, \ldots, c_n\}$ *con* $c_i = \langle \mathsf{con}_i, def_i\rangle$ *y* $def_i : D^{|\mathsf{con}_i|} \to A$.

La contraparte abstracta de P esta definida por el SCSP $\tilde{P} = \langle \tilde{C}, \mathsf{con}\rangle$ sobre el semianillo $\tilde{S}$ en donde:

- $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{\mathbf{0}}, \tilde{\mathbf{1}}\rangle$;
- $\tilde{C} = \{\tilde{c}_0, \ldots, \tilde{c}_n\}$ *con* $\tilde{c}_i = \langle \mathsf{con}_i, \tilde{def}_i\rangle$ *y* $\tilde{def}_i : D^{|\mathsf{con}_i|} \to \tilde{A}$;

- *si* $L = \langle A, \leq_S \rangle$ *es el látice asociado con* S *y* $\tilde{L} = \langle \tilde{A}, \leq_{\tilde{S}} \rangle$ *es el látice asociado con* $\tilde{S}$*, entonces existe una inserción de Galois* $\langle \alpha, \gamma \rangle$ *tal que* $\alpha : L \to \tilde{L}$*;*
- $\tilde{\times}$ *es localmente correcto con respecto a* $\times$*.*

Un aspecto importante en este tipo de abstracción es que la *estructura* del SCSP permanece intacta: tanto C como $\tilde{C}$ tienen las mismas restricciones, y cada una de ellas involucra las mismas variables. La abstracción únicamente cambia el semianillo asociado con el problema de restricciones. De esta forma, P y $\tilde{P}$ tienen la misma topología (en términos de variables y restricciones) con diferentes definiciones de restricción, pues si una tupla de valores para una restricción en P tiene asociado un valor de semianillo v, esa misma tupla (en la misma restricción de $\tilde{P}$) tendrá $\alpha(v)$ como valor asociado en el semianillo. Por otra parte, la definición de la abstracción para un SCSP no condiciona la escogencia de α y γ; únicamente establece que deben satisfacer las propiedades de una inserción de Galois, de las que se derivan (entre otras) que $\alpha(\mathbf{0}) = \tilde{\mathbf{0}}$ y $\alpha(\mathbf{1}) = \tilde{\mathbf{1}}$.

2.2.3.3. Propiedades de la abstracción

A continuación se presentan algunas propiedades del esquema de abstracción definido anteriormente. Algunas de ellas serán utilizadas más adelante en la solución de problemas de satisfacción de restricciones débiles.

Relación entre un SCSP y su versión abstracta Considere el esquema mostrado en la figura 2.3. El látice de problemas concretos está dibujado a la izquierda de la figura, mientras que su contraparte abstracta se ubica a la derecha. El orden parcial en estos látices se representa por líneas punteadas. Sean $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ el semianillo concreto y $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{\mathbf{0}}, \tilde{\mathbf{1}} \rangle$ su versión abstracta, relacionados entre sí por una inserción de Galois $\langle \alpha, \gamma \rangle : \langle A, \leq_S \rangle \rightleftharpoons \langle \tilde{A}, \leq_{\tilde{S}} \rangle$.

Dado un SCSP P, su versión abstracta está definida por $\tilde{P} = \alpha(P)$. Al aplicar la función de concretización γ sobre $\tilde{P}$ se obtiene el nuevo problema $\gamma(\alpha(P))$. Estos dos problemas (P y $\gamma(\alpha(P))$) están relacionados por el siguiente teorema.

Teorema 2.2. *Dado un SCSP* P *sobre* S*, se tiene que* $P \sqsubseteq_S \gamma(\alpha(P))$*.*

Este teorema resulta importante por dos razones. Por una parte, el concepto de *precisión* es formalizado en términos del orden $\leq_S$ asociado con el semianillo. La figura 2.3 representa

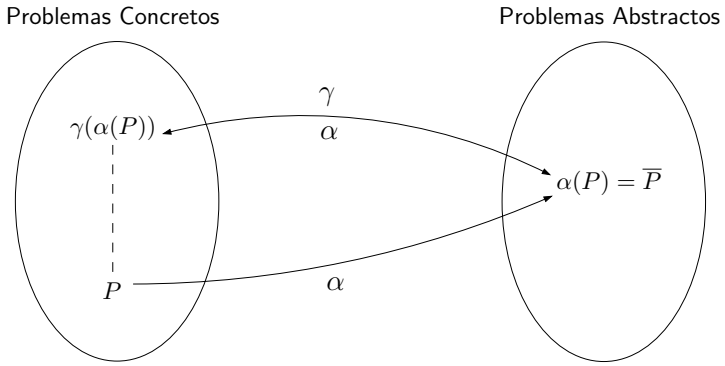

Figura 2.3: Abstracción para SCSPs: problemas concretos y abstractos

gráficamente esta idea: dados dos problemas concretos P y $\gamma(\alpha(P))$, el hecho de que el primero es más preciso que el segundo se representa por su posición vertical en el látice, más cercana al fondo. Por otra parte, del teorema es posible deducir que si una tupla en el problema $\gamma(\alpha(P))$ tiene $\mathbf{0}$ como valor asociado en el semianillo, entonces esa tupla debe tener el mismo valor en P. Más importante aún resulta el saber que este hecho también aplica para las *soluciones* del problema:

Corolario 2.1. *Dado un SCSP P sobre S, se tiene que $Sol(P) \sqsubseteq_S Sol(\gamma(\alpha(P)))$.*

De esta manera se formaliza el hecho de que al pasar de P a $\gamma(\alpha(P))$ no se introducen nuevas inconsistencias: si una tupla en $\gamma(\alpha(P))$ está valuada con $\mathbf{0}$, entonces también está valuada con $\mathbf{0}$ en P. Nótese que algunas inconsistencias en P pueden desaparecer en $\gamma(\alpha(P))$ como resultado del proceso de abstracción.

Ejemplo 2.3. *Considere la Figura 2.4: describe la abstracción del SCSP del Ejemplo 2.1 con uno clásico, siguiendo el esquema de abstracción descrito en la Figura 2.2. En este caso, α transforma en 0 todos los valores concretos en el intervalo $[0, 0.9]$, y en 1 los demás valores. Por su parte, γ convierte el valor abstracto 0 en 0.9 y 1 en 1.*

Si la abstracción preserva el ordenamiento inducido por el semianillo (esto es, si al aplicar la función de abstracción y luego la combinación los elementos están en el mismo orden que se obtendría únicamente de su combinación), existe una relación interesante entre los conjuntos de soluciones óptimas de P y de $\alpha(P)$. De hecho, si una tupla es óptima en P entonces esa misma tupla es también óptima en $\alpha(P)$. Esta propiedad, denominada *preservación del orden,* es definida formalmente a continuación:

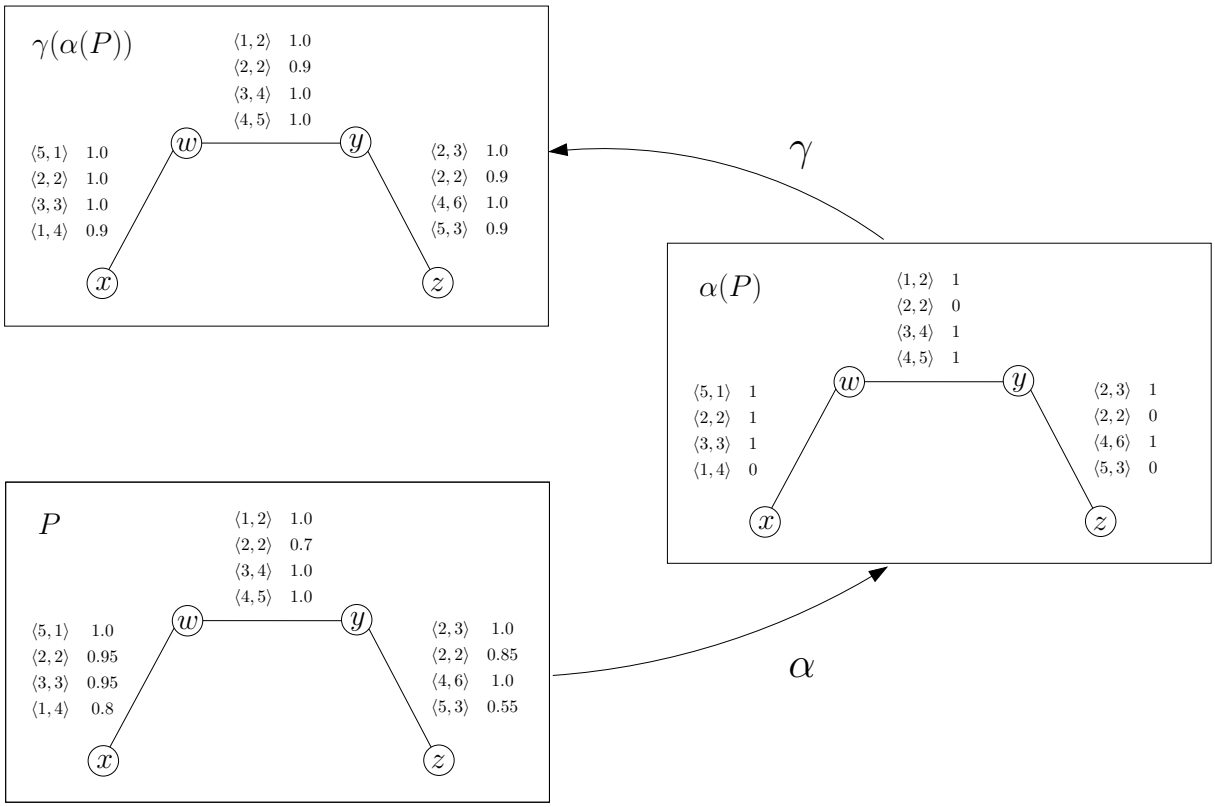

Figura 2.4: Un ejemplo de la abstracción de un SCSP difuso con uno clásico

Definición 2.12 (*Preservación del orden en una abstracción*). *Sean I_1 y I_2 dos conjuntos de elementos concretos. Se dice que una abstracción preserva el orden si*

$$\tilde{\prod_{x \in I_1}} \alpha(x) \leq_{\tilde{S}} \tilde{\prod_{x \in I_2}} \alpha(x) \Rightarrow \prod_{x \in I_1} x \leq_S \prod_{x \in I_2} x$$

en donde las productorias simbolizan las operaciones multiplicativas del semianillo concreto ($\prod$) y del semianillo abstracto ($\tilde{\prod}$).

En otras palabras, si al abstraer y combinar en el dominio abstracto se obtiene el mismo ordenamiento entre los elementos resultantes que el obtenido por la combinación en el dominio concreto, entonces la abstracción preserva el orden.

Ejemplo 2.4. *La abstracción presentada en la figura 2.2 preserva el orden. Considere los únicos dos valores abstractos disponibles (i.e., $0 \leq' 1$) y $\theta = 0.5$. Para obtener $1 = 1 \times' 1 = \alpha(x) \times' \alpha(y)$, x, y deben ser valores concretos por encima de 0.5. Por lo tanto, su combinación concreta (es decir, el mínimo) denotado con v, es siempre mayor que 0.5. Por otro lado, 0 puede obtenerse de la combinación de dos ceros (las imágenes de dos elementos menores o*

iguales que 0.5, cuyo mínimo es menor que 0.5 y por tanto menor también que v), o por la combinación de un 0 y un 1, imágenes de un valor menor que 0.5 y otro mayor que 0.5. En este caso también su combinación (esto es, el mínimo) es menor que 0.5 y por tanto menor que v. Así, la preservación del orden se cumple.

Ejemplo 2.5. *Una abstracción que no preserva el orden es aquella que pasa del semianillo* $\langle N \cup \{+\infty\}, min, sum, 0, +\infty\rangle$*, en donde se minimiza la suma de valores sobre los naturales, al semianillo* $\langle N \cup \{+\infty\}, min, max, 0, +\infty\rangle$*, en donde se busca minimizar el máximo de los valores sobre los naturales. Los ordenamientos de los semianillos son opuestos al orden usual sobre los naturales: si i es menor que j entonces* $j \leq_S i$*, por lo que el mejor elemento es 0 y el peor es* $+\infty$*. Tanto* α *como* γ *son la función identidad.*

En este caso, considere dos valores en el semianillo abstracto y la forma en que son obtenidos, por la combinación de otros dos valores en el mismo semianillo. Por ejemplo, $8 = max(7, 8)$ *y* $9 = max(1, 9)$*. En el ordenamiento abstracto, 8 es mas alto que 9. En el lado concreto,* $sum(7, 8) = 15$ *y* $sum(1, 9) = 10$*, y 15 es menor que 10 en el ordenamiento del semianillo concreto. Así, la propiedad de preservación del orden no se cumple.*

Teorema 2.3. *Considere una abstracción que preserva el orden. Dado un SCSP P sobre S, se cumple que* $Opt(P) \subseteq Opt(\alpha(P))$*.*

Si la abstracción preserva el orden, el conjunto de soluciones óptimas del conjunto abstracto contiene todas las soluciones óptimas del problema concreto. En otras palabras, no hay manera de que una solución óptima en el dominio concreto se convierta en una solución no óptima en el dominio abstracto. No obstante, algunas soluciones concretas no óptimas pueden convertirse en soluciones abstractas óptimas al no poderse distinguir de las soluciones óptimas del problema concreto.

Ejemplo 2.6. *La aplicabilidad del teorema 2.3 puede apreciarse fácilmente considerando las soluciones óptimas de los problemas abstracto y concreto en la figura 2.4:*

$$Opt(\alpha(P)) = \{\langle 5, 1, 2, 3\rangle, \langle 3, 3, 4, 6\rangle\}$$

$$Opt(P) = \{\langle 5, 1, 2, 3\rangle\}.$$

De esta forma, para encontrar una solución óptima para el problema concreto basta con encontrar todas las soluciones del problema abstracto y verificar dichas soluciones en el lado concreto. Esta es una estrategia razonable si se asume que el procesamiento del problema

abstracto es menos complejo que el del problema concreto, pues sólo seria necesario chequear un subconjunto de las tuplas en el problema concreto para obtener soluciones óptimas.

Una propiedad que se cumple para *toda* abstracción, se refiere al cálculo de *cotas* que permitan aproximar las soluciones óptimas en un problema concreto. Cualquier solución óptima t del problema abstracto, valuada por $\tilde{v}$, puede usarse para obtener cotas inferior y superior de un óptimo en P [Bis04]. De hecho, se puede asegurar que existe una solución óptima en P con un valor en el intervalo conformado por el valor de t en P y $\gamma(\tilde{v})$. De este modo, si se considera que es satisfactorio aproximar el valor de una solución óptima con un valor dentro de éste intervalo, se puede tomar t como una aproximación de una solución óptima para P.

Teorema 2.4. *Dado un SCSP P sobre S, sea t una solución óptima para $\alpha(P)$, valuada con $\tilde{v}$ en $\alpha(P)$ y con v en P. Entonces existe una solución óptima $\tilde{t}$ para P (valuada con v') tal que $v \leq v' \leq \gamma(\tilde{v})$.*

Ejemplo 2.7. *En el contexto del ejemplo anterior, considere $\langle 3, 3, 4, 6\rangle$, la solución óptima para $\alpha(P)$ que no es solución óptima de P. $\tilde{v}_1$, la valuación de dicha tupla en el dominio concreto es igual a 0.95, por lo que $\gamma(\tilde{v}_1)$ es igual a 1. De esta forma, se tiene certeza de la existencia de soluciones óptimas en el intervalo $[0.95, 1.0]$, como efectivamente ocurre.*

A partir de los resultados anteriores, dada una tupla t con valor óptimo $\tilde{v}$ en el problema abstracto, en lugar de calcular una solución óptima para P, se puede hacer lo siguiente:

- calcular $\gamma(\tilde{v})$, y obtener así una cota superior del óptimo de P;
- calcular el valor de t en P, cota inferior del mismo óptimo en P;
- si dichas cotas son lo suficientemente cercanas, t puede tomarse como una cota inferior de un óptimo en p.

Como se mencionó anteriormente, esta importante propiedad no requiere que la abstracción preserve el orden, por lo que cualquier abstracción puede aprovecharla.

Procesamiento del problema abstracto A continuación el interés se concentra en el procesamiento del problema abstracto $\alpha(P)$. Nos interesa considerar funciones *intensivas*, es decir, funciones que contribuyan a la solución del problema. Tal contribución se refleja en la posición del problema en el látice, pues resolver un problema implica la combinación de sus restricciones, y por lo tanto "bajar" en el látice. Una de las posibilidades consiste en aplicar una

función $\tilde{f}$, que puede representar, por ejemplo, un algoritmo de consistencia local. Las funciones $\tilde{f}$ deben ser localmente correctas (Definición 2.10) con respecto a alguna función f_{sol} que procese el problema concreto. Esta propiedad se denomina *corrección de las soluciones*. Más precisamente, dado un problema P con un conjunto de restricciones C, $f_{sol}(P)$ es un nuevo problema P' que posee la misma topología que P, con tuplas que tienen valores del semianillo posiblemente menores.

Definición 2.13. *Dado un SCSP P sobre S, sea $\tilde{f}$ una función sobre $\alpha(P)$. Se dice que $\tilde{f}$ es correcta con respecto a las soluciones si, dada alguna f_{sol} que resuelve P, $\tilde{f}$ es localmente correcta con respecto a f_{sol}.*

Una propiedad útil en este caso es la *seguridad* de una función, o la garantía de que una función mantiene todas las soluciones de un problema dado.

Definición 2.14. *Sea PL el conjunto de problemas sobre un semianillo. Dado un SCSP P y una función $f : PL \to PL$, f es segura si $Sol(P) = Sol(f(P))$.*

Al aplicar la función de concretización γ sobre $\tilde{f}(\alpha(P))$, se obtiene $\gamma(\tilde{f}(\alpha(P)))$, problema que se encuentra en el semianillo concreto junto a P. Bajo ciertas condiciones, ilustradas en la figura 2.5, P y $P \otimes \gamma(\tilde{f}(\alpha(P)))$ son equivalentes. Otros hechos pueden apreciarse de dicha figura:

- en el lado abstracto, por las condiciones de intensividad mencionadas antes, $\tilde{f}$ lleva a cualquier elemento más cerca del fondo del látice;
- en el lado concreto, se presenta:
  - por las propiedades de $\otimes$, $P \otimes \gamma(\tilde{f}(\alpha(P)))$ está por debajo tanto de P como de $\gamma(\tilde{f}(\alpha(P)))$ en el látice;
  - por la monotonicidad de γ, $\gamma(\tilde{f}(\alpha(P)))$ está por debajo de $\gamma(\alpha(P))$;
  - por la corrección de las soluciones de $\tilde{f}$, $\gamma(\tilde{f}(\alpha(P)))$ está por encima de $f_{sol}(P)$;
  - por la forma en que es construido, $f_{sol}(P)$ está por debajo de P;
  - si $\times$ es idempotente, entonces debe coincidir con la mayor cota inferior. Por esto, se tiene que $P \otimes \gamma(\tilde{f}(\alpha(P)))$ está por encima de f_{sol}.

Todo lo anterior se formaliza en el siguiente teorema:

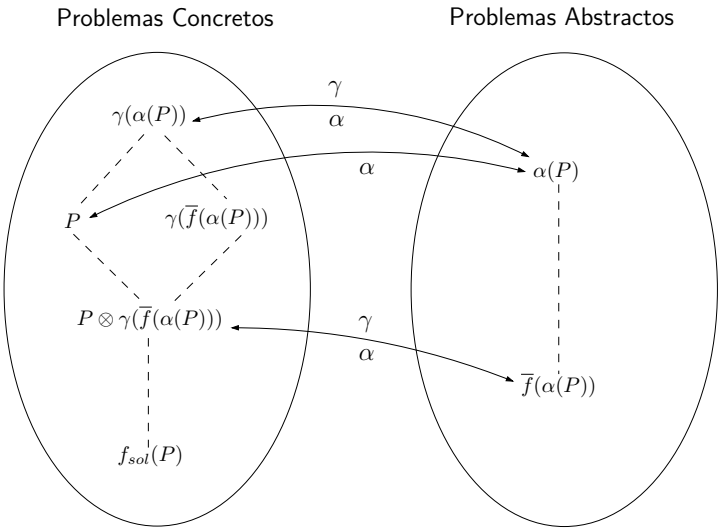

Figura 2.5: Esquema general de abstracción con $\times$ idempotente

Teorema 2.5. *Dado un SCSP P sobre S, considere una función $\tilde{f}$ sobre $\alpha(P)$ que es segura, correcta con respecto a las soluciones e intensiva. Si $\times$ es idempotente, $Sol(P) = Sol(P \otimes \gamma(\tilde{f}(\alpha(P))))$.*

Nótese que el teorema no especifica nada acerca del poder de la función $\tilde{f}$, que puede hacer muchas modificaciones sobre $\alpha(P)$ o incluso no hacer nada. En este ultimo caso, $\gamma(\tilde{f}(\alpha(P))) = \gamma(\alpha(P))$, por lo que $P = P \otimes \gamma(\tilde{f}(\alpha(P)))$, lo que significa que no hubo ninguna ganancia en la abstracción de P. Es posible, sin embargo, usar los teoremas 2.3 y 2.4 para encontrar aproximaciones de las soluciones óptimas y de las inconsistencias de P.

Si por el contrario, $\tilde{f}$ modifica elementos en $\alpha(P)$, y el orden el semianillo concreto es total, se tiene que $P \otimes \gamma(\tilde{f}(\alpha(P))) = \gamma(\tilde{f}(\alpha(P)))$, y por lo tanto se puede trabajar sobre $\gamma(\tilde{f}(\alpha(P)))$ para encontrar las soluciones de P. De hecho, $\gamma(\tilde{f}(\alpha(P)))$ es más bajo (más cercano a las soluciones) que P.

Teorema 2.6. *Dado un SCSP P sobre S, considere una función $\tilde{f}$ sobre $\alpha(P)$ que es segura, correcta con respecto a las soluciones e intensiva. Si $\times$ es idempotente, $\tilde{f}$ modifica todos los elementos del semianillo en $\alpha(P)$, y el orden del semianillo concreto es total, se tiene que $f_{sol}(P) \sqsubseteq_S \gamma(\tilde{f}(\alpha(P))) \sqsubseteq_S P$.*

La idempotencia del operador multiplicativo es necesaria para los teoremas 2.5 y 2.6. Si el operador $\times$ no es idempotente, es posible probar una propiedad menos rigurosa. En tal caso, los problemas $P \otimes \gamma(\tilde{f}(\alpha(P)))$ y $f_{sol}(P)$ no son de interés, pues no tienen las mismas soluciones

que P. Se tiene un nuevo problema P', que resulta de la "inserción" de las inconsistencias de $\gamma(\tilde{f}(\alpha(P)))$ en P. P' es obviamente menor que P en el orden parcial concreto, pues es igual que P con la excepción de algunos ceros, conservando las mismas soluciones.

Teorema 2.7. *Dado un SCSP P sobre S, considere una función $\tilde{f}$ sobre $\alpha(P)$ que es segura, correcta con respecto a las soluciones e intensiva. Si $\times$ no es idempotente, sea P' un SCSP idéntico a P excepto por aquellas tuplas que tienen valuación* $\mathbf{0}$ *en $\gamma(\tilde{f}(\alpha(P)))$: esas tuplas tienen valuación* $\mathbf{0}$ *en P'. Se tiene entonces que $Sol(P) = Sol(P')$.*

En resumen. los teoremas anteriores proveen claves útiles para facilitar la solución de un problema P. Si $\times$ es idempotente, se puede reemplazar por $P \otimes \gamma(\tilde{f}(\alpha(P)))$, y obtener la mismas soluciones (Teorema 2.5). Si no es idempotente, P puede reemplazarse por P' (Teorema 2.7). En cualquier caso, el propósito de estos reemplazos es obtener nuevos problemas (posiblemente más fáciles de resolver que P), pues los valores de semianillos asociados de sus tuplas son más cercanos a los valores de tales tuplas en un problema completamente solucionado.

El razonamiento usado en el Teorema 2.3 sobre $\alpha(P)$ puede aplicarse a $\tilde{f}(\alpha(P))$. De hecho, como $\tilde{f}$ es segura, las soluciones de $\tilde{f}(\alpha(P))$ tienen los mismos valores que aquellas de $\alpha(P)$. De esta forma, los conjuntos de soluciones óptimas coinciden. Por lo tanto, se tiene que $Opt(\tilde{f}(\alpha(P)))$ contiene todas las soluciones óptimas de P si la abstracción preserva el orden. Esto significa que hallando todas las soluciones óptimas de $\tilde{f}(\alpha(P))$ es posible disminuir el espacio de búsqueda requerido para obtener una solución óptima de P.

Teorema 2.8. *Dado un SCSP P sobre S, considere una función $\tilde{f}$ sobre $\alpha(P)$ que es segura, correcta con respecto a las soluciones e intensiva. Si la abstracción preserva el orden, se tiene que $Opt(P) \subseteq Opt(\tilde{f}(\alpha(P)))$.*

2.2.3.4. Una Abstracción Útil

A partir de las instancias de semianillos presentadas en la Sección 2.2.2, pueden concebirse varios esquemas de abstracción. En este trabajo nos interesa analizar la siguiente.

$S_{FCSP} \rightarrow S_{CSP}$ Esta abstracción cambia tanto el dominio como las operaciones del semianillo. La función de abstracción se define por un umbral $\theta \in [0, 1]$, que sirve para asociar los elementos en $[0, \theta)$ con `false` y los elementos en $[\theta, 1]$ con `true`. Por su parte, la función de concretización asocia `true` con 1 y `false` con θ. Un ejemplo de esta abstracción (con $\theta = 0{,}5$)

se muestra en la figura 2.2. Las abstracciones en esta familia preservan el orden (ver Ejemplo 2.4), por lo que el Teorema 2.3 puede aplicarse.

2.3. Resumen

En este capítulo hemos presentado los conceptos y propiedades fundamentales que soportan el presente trabajo. Inicialmente se presentaron las nociones fundamentales del paradigma de programación por restricciones, incluyendo las técnicas de búsqueda y de consistencia. El modelo de programación por restricciones en Mozart fue discutido también, relacionando los aspectos más representativos del modelo con las nociones generales de la programación por restricciones. En segundo lugar, el modelo de restricciones débiles basados en semianillos fue presentado. Las características y propiedades teóricas del modelo, así como instancias particulares del mismo fueron mencionadas. Adicionalmente, un esquema de abstracción para este formalismo, propuesto por Codognet et al. [BCR02], fue descrito detalladamente.

Los conceptos introducidos aquí serán los fundamentos para el desarrollo de componentes prácticos en los capítulos siguientes. En particular, en el capítulo 3 se sentarán las bases para el desarrollo de un módulo de restricciones débiles y de una implementación del esquema de abstracción. Dichas bases comprenden una adaptación orientada a la implementación de los conceptos descritos en este capítulo. La implementación de ambos componentes será discutida en los capítulos 4 y 5.

3 De la Teoría a la Práctica

En este capítulo se presenta una adaptación, orientada a la implementación, del formalismo de restricciones débiles basadas en semianillos y un esquema de abstracción asociado. Inicialmente, se presenta una modificación que acerca las nociones del formalismo a la idea de *propagador*. Dicha modificación define el mecanismo que asocia las tuplas con valores del semianillo (i.e., def) en términos de funciones que calculan dichos valores, evitando su almacenamiento. De esta forma, se establecen las bases para la implementación de restricciones débiles en un lenguaje de programación por restricciones.

En segundo lugar, se definen las características y componentes de un algoritmo iterativo que aprovecha el esquema teórico de abstracción presentado en el capítulo anterior. Dicho algoritmo, basado en el presentado en [BRP03], utiliza un solucionador de restricciones clásicas de dominio finito para emular el comportamiento de restricciones débiles difusas. Las restricciones clásicas que *modelan* restricciones débiles son denominadas *contrapartes fuertes*. Para garantizar la coherencia de los resultados que arrojan las restricciones débiles (surgidas de la modificación mencionada arriba) y estas contrapartes, esta relación de modelamiento es sustentada formalmente.

La modificación al modelo se presenta en la sección 3.1, mientras que la adaptación del esquema de abstracción es discutida en la sección 3.2.

3.1. Restricciones Débiles Basadas en Semianillos Orientadas a la Propagación

Desde la perspectiva de un usuario, puede afirmarse que una desventaja del modelo de restricciones débiles basadas en semianillos es la diferencia existente entre la noción de restricción definida en el modelo y aquella que es establecida en los lenguajes de programación por restricciones. Es decir, para un programador puede resultar engorroso interpretar las condiciones de

un problema como una asignación de valores a tuplas. Por esta razón resulta deseable *adaptar* la noción de restricción dada en el formalismo al concepto de *procedimiento predefinido* que tienen las restricciones en los lenguajes de programación por restricciones. En este contexto, un requerimiento asociado es el de compatibilidad entre las restricciones (clásicas) existentes y el conjunto de restricciones débiles que deben ser incluidas en el lenguaje. Esta cualidad resulta especialmente pertinente en el modelamiento de problemas de la vida real, donde suele ser necesaria la combinación de restricciones fuertes y débiles.

Estos dos requerimientos sugieren modificar los conceptos del formalismo al contexto del modelo de propagación subyacente a la mayoría de los lenguajes de programación por restricciones. Al hacerlo, se establecerían unas bases confiables para la implementación de propagadores débiles. Dichos propagadores débiles quedarían al mismo nivel operativo que sus pares fuertes, asegurando una interacción transparente entre ambos tipos de restricciones.

Para lograr lo anterior, se propone la inclusión en el formalismo de tres nociones que son suficientemente flexibles para modelar relajación en una implementación de restricciones débiles. La primera, denominada *nivel de corte*, define el mínimo nivel de satisfacción aceptado por un usuario en un problema dado; la segunda, llamada *función de consistencia*, es un indicador cuantitativo de la consistencia de una tupla con respecto a una restricción. Finalmente, la *función de valuación* asocia los valores de consistencia con elementos del semianillo, teniendo en cuenta un *factor de penalización* que influye sobre la debilidad asociada con cada restricción. Mientras que el primer concepto está orientado a maximizar la efectividad de los procesos de propagación y distribución, por la interacción entre el segundo y el tercero es posible definir un sistema de valuación de tuplas. Por su concepción, dicho sistema de valuación puede considerarse como una versión más específica de la función def dada en el formalismo original. De esta forma se distinguen en el formalismo dos factores que representan relajación en un SCSP: uno global para el problema y otro local a cada restricción. Dichos factores constituyen una base suficiente para modelar restricciones débiles.

A continuación extendemos y hacemos precisas las ideas dadas arriba. Inicialmente, el nivel de corte para un SCSP puede definirse formalmente de la siguiente manera:

Definición 3.1 (Nivel de Corte). *Dado un semianillo* $S = \langle A, \times, +, \mathbf{0}, \mathbf{1} \rangle$ *y un SCSP* P*, el* nivel de corte $\tau >_S \mathbf{0}$ *es un elemento en* A *que representa el mínimo nivel de consistencia aceptado por el usuario para* P*. Se dice entonces que* P *es consistente para el usuario si y sólo si* $blevel(P) \geq_S \tau$*.*

Esta definición resulta similar a la noción de inconsistencia dada en la definición 2.6, que establece que un problema P es inconsistente si es α-consistente para $\alpha = \mathbf{0}$. No obstante, tal definición resulta poco práctica si se considera que pueden existir soluciones valuadas con valores superiores que $\mathbf{0}$ y que no tienen utilidad alguna para el usuario (e.g., aquellas soluciones con valuaciones muy cercanas a $\mathbf{0}$). La noción de nivel de corte hace preciso el concepto de *solución útil*, capturando de forma explícita la tolerancia del usuario con respecto a la valuaciones asociadas con las soluciones.

Por otra parte, el manejo de las valuaciones asociadas a las tuplas es el principal obstáculo para la definición de procedimientos estándar que implementen restricciones débiles. Es necesario establecer un mecanismo de asociación *por comprensión* entre tuplas y valores del semianillo que reemplace la idea de *enumeración explícita* dada por el formalismo para la definición de una restricción. Dicho mecanismo de asociación puede dividirse en dos partes. La primera obtiene una referencia cuantitativa (i.e. un número natural) entre una tupla dada y una tupla considerada como *correcta* de acuerdo a la restricción que se esté definiendo. Intuitivamente, dicha referencia representa una *distancia* entre ambas tuplas. La segunda parte expresa dicha distancia como un valor del semianillo, completando la asociación deseada para las tuplas. Mientras la primera parte puede entenderse de forma independiente del semianillo utilizado, la segunda debe tener en cuenta el conjunto A de valores del semianillo. Formalmente:

Definición 3.2 (Función de Consistencia). *Sean c una restricción fuerte sobre un conjunto de variables* con *y $\sigma_c : D^{|\mathsf{con}|} \to \mathcal{N}$ una función asociada a ella. Para una tupla $t \in D^{|\mathsf{con}|}$, $\sigma_c(t)$ constituye un indicador cuantitativo de la* inconsistencia *de t con respecto a cualquier tupla consistente con c.*

Nótese que la definición de la función de consistencia es completamente independiente del semianillo usado. Dicha función depende únicamente de la semántica asociada con la restricción c. Diversos criterios pueden tenerse en cuenta para definirla, como eficiencia o nivel de detalle deseado. Como ejemplo considérese la restricción `distinct`, que establece que todos los elementos de una secuencia de enteros deben ser diferentes. Una posible función de distancia para esta restricción es la siguiente:

$$\sigma_{\texttt{distinct}}(t) = nrepeat(t)$$

en donde *nrepeat* retorna el numero de elementos repetidos en una secuencia de enteros. Por ejemplo, $nrepeat([1,1,2]) = 1$ mientras que $nrepeat([1,4,2,7]) = 0$. Sin embargo, pueden haber muchas otras definiciones de consistencia para `distinct` (en [PRB01] se proponen y

estudian algunas otras). Así, una función de consistencia puede considerarse apropiada en la medida en que describa fielmente las características débiles asociadas con una restricción (una especie de "semántica débil"), manteniendo una complejidad razonable.

Para completar la asociación de valores y tuplas, es necesario relacionar los valores retornados por la función de consistencia con elementos del semianillo. En principio, no existen condiciones en la naturaleza de esta relación, por lo que es posible definirla de forma tal que la idea de debilidad dada por la función de consistencia se modifique. En particular, tal mecanismo de asociación puede incluir un *factor* ligado con cada restricción. Dicho factor puede influir directamente en la asociación final entre tuplas y elementos del semianillo. Formalmente:

Definición 3.3 (Función de Valuación). *Considere una función de consistencia σ_c para una restricción fuerte c. Una función de valuación $\rho_\beta : \mathcal{N} \to A$ asocia todo valor dado por σ_c con un elemento en A, parametrizada por un factor $\beta \in A$. β es denominado el* factor de penalización *asociado con c.*

En síntesis, toda restricción c depende de dos funciones complementarias para valuar las tuplas: σ_c y ρ_β. Estas dos nociones se engloban en la siguiente definición:

Definición 3.4 (Sistema de Penalización). *Dada una restricción fuerte c, un sistema de penalización $\langle \sigma_c, \rho_\beta \rangle$ para c está conformado por una función de consistencia σ_c y una función de valuación ρ_β.*

Con las definiciones anteriores es posible redefinir el concepto de restricción del modelo formal de semianillos:

Definición 3.5 (Restricción Débil). *Sean c una restricción fuerte sobre un conjunto de variables con, un sistema de restricciones $CS = \langle S, D, V \rangle$ (con $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$) y un sistema de penalización $\langle \sigma_c, \rho_\beta \rangle$ para c. La versión débil de la restricción c es una pareja $\langle def_{\langle \sigma_c, \rho_\beta \rangle}, \mathsf{con} \rangle$ en donde:*

1. $\mathsf{con} \subseteq V$*, es denominado el* tipo *de la restricción;*
2. $def_{\langle \sigma_c, \rho_\beta \rangle} : D^{|\mathsf{con}|} \to A$*, tal que* $def_{\langle \sigma_c, \rho_\beta \rangle}(t) = \rho_\beta(\sigma_c(t))$*, para toda tupla* $t \in D^{|\mathsf{con}|}$.

Un aspecto fundamental de esta noción de restricción es que es totalmente ortogonal con respecto a las definiciones originales de restricción y de problema de restricciones (Definición 2.3): ambas definiciones asocian inequivocamente tuplas de valores con elementos del

semianillo. Nuestra definición hace más detallada esta asociación, sin que dicha información adicional constituya una limitación u ocasione una pérdida de generalidad en el formalismo. Por el contrario, nuestro esquema de asociación puede ser fácilmente parametrizado, abriendo un amplio abanico de posibilidades para la definición de restricciones débiles. El siguiente ejemplo ilustra estas ideas.

Ejemplo 3.1 (Una restricción débil). *Considere la restricción fuerte $X < Y$. Una versión débil de esta restricción para el semianillo difuso puede definirse como $c = \langle\{X, Y\}, def_{\langle\sigma_{lt},\rho_{0.2}\rangle}\rangle$, en donde:*

- $\sigma_{\mathtt{lt}}(i, j) = \mathtt{max}(i - j + 1.0\ ,\ 0.0), \quad \forall_{\langle i,j\rangle} \in dom(X) \times dom(Y)$
- $\rho_{0.2}(x) = \mathtt{max}(0.0, 1 - (x * 0.2))$.

Intuitivamente, la función de consistencia $\sigma_{\mathtt{lt}}$ retorna el numero de "unidades" en que c es violada. Nótese que en el caso en que la restricción se cumpla totalmente, $\sigma_{\mathtt{lt}}$ retornará 0. La función $\rho_{0.2}$ traslada estas unidades al contexto de números reales entre 0 y 1. De esta forma, las valuaciones asociadas con las tuplas $\langle 4, 4\rangle$ y $\langle 5, 4\rangle$ son $def(\langle 4, 4\rangle) = 1.0 - 0.2 \times (1.0) = 0.8$ y $def(\langle 5, 4\rangle) = 0.6$.

Resulta interesante tener una idea concreta de los efectos de una restricción débil en el contexto particular de un SCSP P. Específicamente, dichos efectos se pueden entender como la *tolerancia* que la restricción tiene con respecto a la posible inconsistencia de una tupla particular. Esta idea, que depende del nivel de corte τ asociado con P, se formaliza en la siguiente definición.

Definición 3.6 (Tolerancia de una restricción débil). *Sea un SCSP $P = \langle C, \mathsf{con}\rangle$ y τ su nivel de corte asociado. Para una restricción débil $c_d \in C$, $tol(c_d)$ su* tolerancia *con respecto a P, es una función $tol : C \to \mathcal{N}$ definida por*

$$tol(c_d) = \left\lfloor \frac{\mathbf{1} - \tau}{\beta_{c_d}} \right\rfloor,$$

donde β_{c_d} es el factor de penalización asociado con c_d. Intuitivamente, $tol(c_d)$ es un número natural que cuantifica las violaciones permitidas por c_d.

La tolerancia de una restricción representa una referencia para las valuaciones de las tuplas. Se trata de una representación directa de los deseos del usuario, que son expresados en términos

de β y de τ. Por definición, la tolerancia de una restricción es completamente dependiente del problema.

De las definiciones anteriores es posible apreciar cómo el factor de penalización β y el nivel de corte τ proporcionan un control directo de las valuaciones asociadas a las tuplas y de la calidad de las soluciones deseadas, respectivamente. Este esquema es conveniente para una implementación, pues no hay necesidad de almacenar y procesar datos relacionados con las valuaciones; las funciones de consistencia y valuación así como el factor de penalización están asociados con cada restricción, mientras que el nivel de corte rige para todo el problema de restricciones. Con estas dos nociones es posible definir el concepto de función de filtrado para un propagador débil:

Definición 3.7 (Función de filtrado). *Dada una restricción débil* $c_d = \langle def_{\langle \sigma_c, \rho_\beta \rangle}, \mathsf{con} \rangle$, *una función de filtrado para* c_d *(denotada por* F_{c_d}*) es un operador de acortamiento [BMH94] que para toda variable* $x_i \in \mathsf{con}$ *y toda tupla* $t \in D^{|\mathsf{con}|}$ *descarta aquellos valores* $d_i \in dom(x_i)$ *tales que* $t \downarrow_{x_i} = d_i$, $def_{\langle \sigma_c, \rho_\beta \rangle}(t) <_s \tau$.

Una función de filtrado para una restricción débil se concentra en remover todos aquellos valores de las variables que hacen parte de tuplas valuadas con valores del semianillo menores que τ. Una interpretación equivalente consiste en pensar que una función de filtrado se acoge a la tolerancia de la restricción débil y descarta aquellos valores que la superan. De esta forma, la tolerancia del usuario a la violación de una restricción se convierte en el criterio más importante en la definición de propagadores: de ella dependen no sólo la cantidad de soluciones que se obtienen, sino también los recursos requeridos para encontrarlas, pues por ejemplo, valores de τ muy cercanos a $\mathbf{0}$ (un valor de tolerancia alto) tienen asociado un espacio de búsqueda más amplio que el espacio asociado con valores para τ cercanos a $\mathbf{1}$ (poca tolerancia a violaciones).

Con las definiciones anteriores, es posible definir la noción de un propagador débil:

Definición 3.8 (Propagador Débil). *Dada una restricción débil* $c_d = \langle def_{\langle \sigma_{c_d}, \rho_\beta \rangle}, \mathsf{con} \rangle$, *un* propagador débil *para* c_d *es un agente computacional encargado de aplicar una función de filtrado* F_c.

Con esta definición se completa el proceso de adaptación del modelo formal de restricciones débiles basado en semianillos. La noción de propagador, como elemento que implementa una restricción de filtrado de valores, es común en casi todas las implementaciones de lenguajes de programación por restricciones.

3.2. Un Esquema Iterativo de Solución para SCSPs

Encontrar una solución (óptima) para un SCSP usando la modificación propuesta anteriormente puede resultar costoso en términos computacionales. La relajación de las restricciones en un problema implica una reducción en el poder de las funciones de filtrado asociadas con las restricciones, aumentando así el espacio de búsqueda que debe ser considerado en los procesos de distribución. Los costos asociados con la inclusión de restricciones débiles en problemas medianos o grandes son generalmente significativos, por lo que encontrar mecanismos efectivos que permitan la solución de este tipo de problemas es imperativo.

En este contexto, el esquema de abstracción para SCSPs descrito en la sección 2.2.3 constituye una alternativa válida para encontrar y/o aproximar soluciones óptimas para problemas de restricciones débiles en un tiempo razonable. La idea es procesar el problema abstracto y extraer *información útil* de ese proceso para facilitar la solución del problema concreto. Se sabe que dicha información útil consiste básicamente de las soluciones óptimas del problema abstracto: se trata de indicadores del nivel de satisfacción general del problema que pueden ser soluciones óptimas del problema concreto. En esta sección nos interesa explotar estas características de las soluciones del problema abstracto, en términos de un algoritmo iterativo que ubica aquellas soluciones del problema con valuaciones más altas.

En este trabajo nos concentramos en el esquema de abstracción que contribuye a la solución de un problema difuso usando un solucionador de restricciones clásico. Esta decisión no resta generalidad a los resultados descritos a continuación; es posible aplicarlos a cualquier esquema de abstracción que preserve el orden (Definición 2.12). La ventaja principal de dicho esquema de abstracción (además de las propiedades discutidas en el capítulo anterior) consiste en que aprovecha la existencia de un sistema de restricciones fuertes en los procesos de abstracción. De esta forma, cualquier lenguaje de programación que provea restricciones de forma explícita puede implementar técnicas de restricciones débiles.

El algoritmo básico que apoya el esquema iterativo de solución incluye los siguientes procesos:

- abstracción de un SCSP difuso en uno clásico (implementando la función α de una inserción de Galois), utilizando *contrapartes fuertes* que emulan restricciones débiles difusas, y
- procesamiento sistemático de dicho problema abstracto.

En algunos casos, el algoritmo hace uso en el dominio concreto de la información obtenida en el abstracto (implementando la función γ de la inserción de Galois).

Cada una de estas fases es descrita a continuación. Luego, la relación formal existente entre las contrapartes fuertes del esquema de abstracción y los propagadores débiles de la sección anterior es descrita.

3.2.1. Algoritmo Básico

El algoritmo 3.1 representa el esquema básico de búsqueda de soluciones óptimas basado en la abstracción. El algoritmo busca la mejor solución para el problema concreto, con base en las respuestas de un solucionador para el problema abstracto. Para esto, el algoritmo conserva un *intervalo* de valores del dominio concreto, que representa el rango en el que se encuentra dicha solución. Los parámetros de entrada del algoritmo son un SCSP P, un valor de precisión `Delta`, un intervalo Inter y una solución `MejorSol`. Éstos dos últimos representan los valores actuales del intervalo mencionado antes y de la mejor solución obtenida hasta el momento, respectivamente. Se asume que tal solución tiene asociada su respectiva valuación concreta.

Una ejecución del algoritmo comienza por la verificación del tamaño del intervalo de entrada. Si dicho tamaño es menor que la precisión dada (`Delta`), el proceso retorna la mejor solución y termina. De lo contrario, el algoritmo busca una solución en la versión abstracta de P, tomando como nivel de corte en dicha búsqueda el límite inferior del intervalo de entrada. Con el resultado de dicho proceso de solución, un nuevo intervalo es generado. Si el solucionador abstracto obtuvo una solución, se invoca una llamada recursiva del algoritmo, tomando la solución obtenida como nueva mejor solución. De lo contrario (el solucionador abstracto no retornó solución, denotado por `empty`), se invoca una llamada recursiva del algoritmo manteniendo el valor de entrada para la mejor solución. La terminación del algoritmo está determinada por la función GenereNuevoIntervalo que debe retornar un intervalo de tamaño menor que el intervalo de entrada, independientemente de la respuesta del solucionador abstracto.

El algoritmo 3.1 es completamente independiente del esquema de abstracción. A continuación se describen las características de las funciones ResuelvaAbstracto y GenereNuevoIntervalo para el esquema de abstracción particular que nos interesa analizar aquí. Más adelante, el rol de la función de concretización en el algoritmo 3.1 será descrito detalladamente.

Algoritmo 3.1 Algoritmo Iterativo para Resolver SCSPs

SolucionIterativa := **proc** (P, `Delta`, Inter, `MejorSol`)

```
1: if AlcanzoPrecision?(Delta, Inter) = true then
2:    retorne MejorSol
3: else
4:    Sol = ResuelvaAbstracto(P, Inter.low)
5:    NuevoInter = GenereNuevoIntervalo(Inter, Sol)
6:    if Sol ≠ empty then
7:       SolucionIterativa(P, Delta, NuevoInter, Sol)
8:    else
9:       SolucionIterativa(P, Delta, NuevoInter, MejorSol)
```

3.2.1.1. Solución de un Problema Abstracto

El objetivo de la función de abstracción α es convertir un CSP difuso en uno clásico, sin afectar la estructura del problema original. Es necesario entonces definir una conversión entre los valores del semianillo del CSP difuso (números reales entre 0 y 1) y los dos únicos valores asociados con un problema clásico (1 y 0). Una posible definición para α consiste en asociar todos aquellos valores del semianillo superiores a un umbral θ con 1, y los valores iguales o inferiores que θ con 0. La figura 2.2 representa dicho esquema, para $\theta = 0{,}5$.

En nuestro caso, y debido a la modificación al formalismo presentada anteriormente, la manipulación explícita de las valuaciones asociadas con las tuplas no es una opción valida. Por lo tanto, es necesario *modelar* el comportamiento de las restricciones difusas usando las restricciones fuertes disponibles y los valores β y τ (tomados del dominio concreto). Esto se puede lograr igualando el umbral θ con el nivel de corte τ. El objetivo final de dicho modelamiento es asegurar que los dos problemas, el difuso y el clásico, acepten las mismas tuplas como soluciones.

Para realizar de manera automática la conversión definida por α, se definen restricciones clásicas especiales para cada restricción difusa. Dichas restricciones clásicas se caracterizan por incluir un parámetro adicional que induce relajación, denominado *valor de holgura*. Dicho parámetro es calculado de forma independiente para cada restricción teniendo en cuenta β, τ y 1, el valor de aceptación para el semianillo clásico. Estas restricciones clásicas —parametrizadas por los valores de holgura— son denominadas *contrapartes fuertes* de sus restricciones difusas análogas. De esta forma, el objetivo de α es calcular los valores de holgura correspondientes para cada contraparte fuerte de una restricción difusa, con la seguridad de que en ambos dominios (el abstracto y el concreto) las mismas tuplas son aceptadas como

solución.

Como en el caso de las funciones de consistencia, la definición exacta de una contraparte fuerte depende exclusivamente de la semántica de la restricción difusa que se desea modelar. En el caso de las restricciones aritméticas, una regla general para generar las contrapartes fuertes consiste en relajar las (des)igualdades incluidas en ellas. Esto se logra reemplazando las igualdades por desigualdades e incluyendo valores de holgura apropiados en las expresiones matemáticas que contienen desigualdades. Este simple criterio es válido para un amplio rango de restricciones, desde expresiones como $X \leq Y$ hasta restricciones que definen polinomios. Para otros tipos de restricciones (no aritméticas), el proceso de definición de contrapartes fuertes se asemeja mucho a la definición de funciones de consistencia (Definición 3.2), en donde diversos criterios o factores pueden inducir diferentes definiciones de contrapartes fuertes. El siguiente ejemplo ilustra estas ideas.

Ejemplo 3.2. *Considere las restricciones difusas $Exp_1 < Exp_2$ y $Exp_1 + Exp_2 = Exp_3$, en donde cada Exp_i es una expresión aritmética. La función α calcula los valores de holgura correspondientes que junto con las contrapartes fuertes son usados para convertir un CSP difuso en su equivalente clásico. Por ejemplo, la contraparte clásica para la restricción menor o igual son las siguientes:*

$$Exp_1 - H_1 < Exp_2. \tag{3.1}$$

Todas las tuplas aceptadas por la restricción menor o igual son aceptadas también por su contraparte clásica asociada. Informalmente, suponga $Exp_1 = X$ y $Exp_2 = Y$, dos variables de dominio finito. Asuma $\tau = 0{,}8$ y $\beta = 0{,}05$. El valor de holgura (denotado en la expresión 3.1 por H_1) se obtiene usando la siguiente expresión:

$$H_1 = \frac{1 - \tau}{\beta}.$$

Informalmente, el valor de holgura representa el número de "unidades" en las que se puede violar la restricción (que se obtiene considerando τ y 1, la máxima valuación permitida), sujeto al valor concreto de β. En este caso, $H_1 = 4$. La tupla $\langle X = 3, Y = 2\rangle$, valuada en el dominio concreto con 0.9, es aceptada por la contraparte fuerte definida arriba pues la desigualdad se cumple ($3 - 4 < 2$). Por el contrario, la tupla $\langle X = 7, Y = 1\rangle$, rechazada en el dominio concreto, es también rechazada pues la desigualdad se incumple (i.e., $7 - 4 \not< 1$).

Por otra parte, la contraparte fuerte para la restricción de suma está definida por la siguiente expresión (se asume $Exp_3 = Z$):

$$\texttt{abs}((Exp_1 + Exp_2) - Exp_3) \leq H_2, \tag{3.2}$$

donde, nuevamente, el valor exacto de H_2 depende del β asociado con la restricción.

Estas intuiciones con respecto a la equivalencia de las restricciones débiles y las contrapartes fuertes serán formalizadas en la sección 3.2.2.

3.2.1.2. El Rol de la Función de Concretización

En el esquema de abstracción discutido en el capítulo anterior, la función de concretización γ traslada un problema abstracto $\alpha(P)$ al dominio concreto, modificando las valuaciones asociadas con las tuplas para las restricciones en P. En el algoritmo 3.1, sin embargo, su rol es menos determinante que el de la función de abstracción α. En dicho contexto, el papel de la función de concretización se reduce a proveer la valuación concreta asociada con una solución abstracta. Esta función es explícitamente invocada cuando ResuelvaAbstracto encuentra una solución, de forma que la solución y su valuación concreta puedan considerarse de forma conjunta. Adicionalmente, el valor de la valuación concreta puede aprovecharse en la generación de nuevos intervalos para el algoritmo iterativo, como se describe más adelante.

Operacionalmente, para obtener la valuación concreta se consideran las nociones originales del formalismo, en lo concerniente a la combinación de las valuaciones asociadas con las tuplas.

Ejemplo 3.3. *Considere las dos restricciones del ejemplo anterior junto con la tupla $\langle X = 4, Y = 3, Z = 5\rangle$ (que es inconsistente para ambas), asumiendo un $\beta = 0.07$ para la segunda restricción. Para dicha tupla, la función γ retorna una valuación de 0.86 (1.0 - 0.14) en el dominio concreto. Dicha valuación se obtiene de considerar 0.14, la penalización inducida por ambas restricciones. A su vez, dicha penalización se obtiene al considerar la máxima violación entre la penalización inducida por la restricción de la suma (0.07*2=0.14) y la penalización inducida por la restricción menor que (0.05*2=0.1).*

3.2.1.3. Generación de Intervalos

Como se dijo anteriormente, en la generación de un nuevo intervalo para el algoritmo iterativo debe tenerse en cuenta el resultado del proceso de solución sobre el problema abstracto. Un aspecto fundamental en este algoritmo es que el nivel de corte usado en dicha búsqueda es igual al límite inferior del intervalo. Esto hace que las estrategias para generar nuevos intervalos estén orientadas a encontrar el mejor límite inferior posible (esto es, el más cercano a 1), de modo que los procesos de solución posteriores sean lo más efectivos posibles. A continuación

definimos tres posibles estrategias de generación de nuevos intervalos para el algoritmo 3.1. En dichas estrategias, t representa el mínimo nivel de satisfacción aceptado por el usuario.

1. *Estrategia Binaria.* Esta estrategia de selección de nuevos intervalos está inspirada en una búsqueda binaria. El intervalo inicial está dado por un parámetro dado por el usuario y 1, el mayor valor posible en semianillo difuso. Dado un intervalo $[l, u]$, si después del proceso de solución sobre el problema abstracto se encuentra una solución, el nuevo intervalo será igual a $[(l + u)/2, u]$. En caso contrario, el intervalo generado será $[b, l]$, en donde b es calculado teniendo en cuenta tanto los lineamientos de la búsqueda binaria como la información del ultimo intervalo en donde se encontró una solución. Esta estrategia está dispuesta de tal modo que las precauciones necesarias para no repetir procesos de búsqueda con el mismo nivel de corte han sido consideradas.

2. *Estrategia Ávida.* En esta estrategia se aprovecha la información de la función de concretización γ para mejorar el límite inferior del intervalo dado por la estrategia binaria. Cuando hay una solución, el valor de dicho límite será el máximo entre la valuación de la solución encontrada en el dominio concreto (obtenida usando γ) y el límite inferior dado por el modo binario. Consecuentemente, esta estrategia puede considerarse como una especialización de la estrategia binaria, obteniendo idénticos resultados en el caso en que el solucionador abstracto no obtenga solución.

3. *Estrategia Pesimista.* A diferencia de las estrategias discutidas anteriormente, esta estrategia está orientada a aquellos casos que no tienen solución o cuya solución tiene una valuación muy cercana a 0. Dado un intervalo inicial $[l, u]$, si el solucionador abstracto no encuentra solución el segundo intervalo generado será igual a $[t, l]$. A partir de ese punto, la ejecución continúa bajo los lineamientos del modo binario. Esto sugiere una elección cuidadosa del valor de t, de modo que si hay solución en el intervalo $[t, l]$ el modo binario subsiguiente sea tan eficiente como sea posible. Como en la estrategia ávida, si el pesimismo del usuario es infundado, la estrategia pesimista se comportará igual que la estrategia binaria.

3.2.2. Relación entre un Propagador Débil y su Contraparte Fuerte

Es fundamental que los efectos de un propagador débil sean los mismos que los de las contrapartes fuertes presentadas arriba. Este tipo de equivalencia confiere suficiente seguridad al momento de comparar los resultados obtenidos por uno u otro mecanismo. A continuación

esta propiedad es demostrada para el propagador relacional $X < Y$, siguiendo un esquema por contradicción que puede ser aplicado similarmente en los demás propagadores/contrapartes relacionales fuertes. Informalmente, el propósito de la prueba es mostrar cómo una contraparte fuerte se comporta de la misma manera que las reglas de propagación de su correspondiente propagador débil. La prueba se basa en el hecho de que la mejor manera de describir el comportamiento de un propagador es a través de su *regla de propagación*. A modo de convención, el propagador de la relación $<$, será denotado por $<:$.

Teorema 3.1. *Para la restricción $X < Y$, tanto el propagador débil como su contraparte fuerte tienen los mismos efectos.*

PRUEBA. Inicialmente, es necesario relacionar los elementos que permiten relajación de uno y otro mecanismo, el incremento usado en las reglas de propagación de los propagadores débiles y el valor de holgura en las contrapartes fuertes:

Suposición 3.1. *El valor de tolerancia de un propagador débil (denotado con m) es igual al valor de holgura de las contrapartes fuertes (denotado con h).*

Se asumen variables de dominio finito denotadas con $X, Y...$, en donde los límites superior e inferior del dominio para X son denotados por $\overline{x}$ y $\underline{x}$, respectivamente. Adicionalmente, los efectos de las reglas de propagación débiles sobre dos variables de dominio finito de entrada X_{in} y Y_{in} se denotan con X_{out} y Y_{out}, respectivamente. Dichos efectos, usando las reglas asociadas con la contraparte fuerte, se denotan con las variables X'_{out} y Y'_{out}. De esta forma, la hipótesis a contradecir es que $X_{out} \neq X'_{out}$ y que $Y_{out} \neq Y'_{out}$. Las reglas de propagación para el propagador débil $X < Y$ se describen a continuación:

$$X \leq_* \overline{y} + m - 1 \tag{3.3}$$

$$Y \geq_* \underline{x} - m + 1 \tag{3.4}$$

en donde $X \geq_* (\leq_*) y_i$ denota una operación que remueve todos los elementos menores (mayores) que el entero y_i del dominio X . Por su parte, la contraparte fuerte está definida por la siguiente restricción:

$$X <: Y + h, \tag{3.5}$$

que puede expresarse también como

$$X - h <: Y. \tag{3.6}$$

Por su parte, la restricción fuerte que impone la relación menor que (<:) está definida de la siguiente forma:

$$X \leq_* \overline{y} - 1 \tag{3.7}$$

$$Y \geq_* \underline{x} + 1. \tag{3.8}$$

La estrategia de prueba consiste en suponer que efectivamente sí existen diferencias entre el resultado de aplicar las reglas de propagación débil y las reglas de propagación asociadas con la contraparte fuerte. Suponga dos variables de dominio finito $X_{in} = [\underline{x}, \overline{x}]$ y $Y_{in} = [\underline{y}, \overline{y}]$ que representan el estado de las variables X y Y *antes* del proceso de propagación. Aplicando las reglas 3.3 y 3.4 (correspondientes a los propagadores débiles) se tienen las siguientes relaciones:

$$X_{in} \leq_* \overline{y} + m - 1 \tag{3.9}$$

$$Y_{in} \geq_* \underline{x} - m + 1. \tag{3.10}$$

Expresada como dominios finitos, la restricción asociada con la contraparte fuerte (i.e., 3.5 o 3.6) para dichas variables de entrada es la siguiente:

$$X_{in} <: Y_{in} + [0, h] \quad (\text{o, alternativamente, } [\underline{x}, \overline{x}] + [-h, 0] <: Y_{in}),$$

que puede simplificarse en la siguiente expresión:

$$X_{in} <: [\underline{y}, \overline{y} + h] \quad (\text{o, alternativamente, } [\underline{x} - h, \overline{x}] <: Y_{in}). \tag{3.11}$$

Aplicando 3.7 y 3.8 sobre los dominios definidos en 3.11 se tienen las siguientes reglas de propagación:

$$X \leq_* \overline{y} + h - 1 \tag{3.12}$$

$$Y \geq_* \underline{x} - h + 1. \tag{3.13}$$

En este punto es claro que la operaciones que finalmente deciden los efectos sobre los dominios finitos de entrada son $\leq_*$ y $\geq_*$. Comparando las relaciones 3.9 y 3.10 con 3.12 y 3.13, y respetando la hipótesis inicial debe cumplirse que:

$$\overline{y} + h - 1 \neq \overline{y} + m - 1 \tag{3.14}$$

$$\underline{x} - h + 1 \neq \underline{x} - m + 1. \tag{3.15}$$

Claramente, para que 3.14 y 3.15 se mantengan resulta indispensable que $h \neq m$, lo que es falso de acuerdo a la suposición 3.1. De esta forma, el teorema queda demostrado.

3.3. Discusión

En las extensiones presentadas en este capítulo dos aspectos son especialmente sobresalientes. El primero de ellos tiene que ver con la *conservatividad* que ellas persiguen, mientras que el otro tiene que ver con la *especificidad* de sus propósitos.

En general, puede afirmarse que tanto la noción de propagador relacionada con la definición de restricción débil como los elementos asociados con el algoritmo iterativo de abstracción son conservativos con respecto a las propuestas teóricas originales. En primer lugar, la noción de propagador propuesta es completamente ortogonal con respecto a la noción original de restricción dada en el formalismo (Definición 2.3); ella puede considerarse como una especificación de la asociación entre restricciones y valores del semianillo. Las funciones de consistencia y valuación son tan sólo un instrumento para articular una asociación orientada a una implementación económica; el nivel de corte y los factores de penalización ayudan a precisar dicha asociación. La idea de contraparte fuerte, nuestra contribución al esquema iterativo de abstracción, sigue esta filosofía de conservatividad al constituirse como una alternativa concreta de solución de problemas abstractos clásicos. De esta forma, la integración de esta idea no interfiere ni contradice los fundamentos del esquema de abstracción original. Consecuentemente, el algoritmo 3.1 resulta una interpretación particular del propuesto en [BCR02] que incluye las nociones de las contrapartes fuertes.

En términos de especificidad, las extensiones se concentran en aprovechar aquellos resultados que aplican para semianillos con operadores multiplicativos idempotentes. Esta decisión garantiza un amplio espectro de resultados y propiedades que pueden ser aprovechadas en una implementación, y guarda concordancia con el estado del arte, en donde el manejo eficiente de CSPs débiles con operadores no idempotentes es aún un problema abierto. Intuitivamente, la dificultad de solucionar este tipo de problemas se debe a que los métodos clásicos de mantenimiento de consistencia local resultan incorrectos. Aunque recientemente se han propuesto algunas soluciones teóricas para este problema ([CS04, GSZ05]), a nuestro mejor saber, aun no se han publicado trabajos que las implementen apropiadamente.

3.4. Resumen

En este capítulo se han presentado adaptaciones —orientadas a la implementación— del modelo de restricciones débiles basado en semianillos y del esquema de abstracción propuesto

por Codognet et al. ([BCR02]) para dicho modelo. Para el primero, una noción parametrizable de asociación de tuplas y valores del semianillo ha sido propuesta. Dicha noción define dicha asociación en términos de una función de consistencia y otra de valuación. Ambas funciones constituyen un esquema económico de valuación de tuplas, que puede parametrizarse por medio de un factor de penalización. Esta noción, que puede entenderse entonces como una *especialización* de la función def que compone cada restricción, constituye una modificación ortogonal del concepto de restricción débil que es apropiada para una implementación.

Por otra parte, en este capítulo se propuso también un algoritmo iterativo que aprovecha las propiedades teóricas del esquema de abstracción. Dicho algoritmo, que guarda semejanza con respecto al presentado en [BRP03], tiene como objeto encontrar el nivel de corte más alto para un SCSP. Para lograrlo, el algoritmo depende de unas contrapartes (definidas en términos de restricciones fuertes) que modelan restricciones difusas. Adicionalmente, el algoritmo puede parametrizarse por medio de funciones (de generación de intervalos) que pueden adaptarse a varias situaciones.

Es conveniente destacar que por la manera en que ambas modificaciones están dadas, es posible desarrollar una implementación de alguna de ellas en cualquier lenguaje de programación por restricciones en el que la noción de propagador esté presente y que posea un módulo de restricciones clásicas de dominio finito. En los siguientes capítulos, el reto de obtener una idea práctica de ambas adaptaciones será abordado en el contexto de Mozart.

Algunos de los resultados de este capítulo han sido publicados previamente. Una versión inicial de la modificación del concepto de restricción fue presentada en [DOPR05b]. El algoritmo iterativo para el esquema de abstracción así como una descripción de la noción de contraparte fuerte fueron presentadas en [DPR05]. Los resultados descritos en este capítulo constituyen una versión ampliada de dichos trabajos. Finalmente, la relación formal entre los propagadores débiles y las contrapartes fuertes es original a este trabajo de grado.

4 Un Solucionador para CSPs Difusos

Este capítulo presenta un módulo de restricciones débiles para Mozart basado en la modificación conceptual al concepto de *propagador* propuesta en el capítulo anterior. En particular, el módulo provee operaciones para modelar y solucionar problemas de satisfacción de restricciones *difusos*. Como se recordará, en este tipo de problemas se asocian valores reales entre 0 y 1 con cada una de las tuplas que representan las restricciones. El objetivo es encontrar asignaciones de valores a variables que exhiban la máxima satisfacción global posible.

Los componentes de la implementación pueden dividirse en dos categorías de acuerdo a los procesos principales de solución en Mozart: un conjunto de propagadores débiles que siguen los lineamientos establecidos en el capítulo 3 y una serie de criterios para distribución específicos para CSPs difusos. Mientras que los primeros se sirven de los mecanismos de extensión que Mozart provee para integrar transparentemente nuevas restricciones en el lenguaje, los segundos explotan los elementos característicos del módulo de restricciones débiles para la implementación de criterios más precisos de selección de variables.

El capítulo se organiza de acuerdo a estas dos contribuciones. La sección 4.1 se ocupa de describir de manera detallada los propagadores débiles implementados, relacionándolos con el modelo formal modificado. La sección 4.2 describe los criterios de distribución mencionados y justifica sus fundamentos. Casos de estudio que evalúan los componentes implementados en problemas de la vida real son presentados en la sección 4.3. Por ultimo, la discusión presentada en la sección 4.4 argumenta las diferencias entre la implementación de semianillos con operadores multiplicativos idempotentes y aquellos que no satisfacen esta propiedad.

4.1. Elementos de Propagación

Los elementos de propagación se dividen en tres categorías: restricciones relacionales, aritméticas y globales. Para cada una de ellas se proporciona su función de consistencia y se

describe su sintaxis en Mozart. Todas las restricciones comparten la siguiente función de valuación:

$$\rho(j) = \texttt{max}(0.0\ , 1 - (j \times \beta)),$$

para un número entero j y un factor de penalización $\beta \in (0, 1)$.

Restricciones Relacionales para los operadores en el conjunto $RelOp = \{<, \leq, >, \geq\}$. Dadas dos variables de dominio finito X y Y, estas restricciones están definidas por las siguientes funciones de consistencia ($\langle d_i, d_j \rangle \in dom(X) \times dom(Y)$):

- $\sigma_{less}(\langle d_1, d_2 \rangle) = \texttt{max}(0, d_1 - d_2 + 1)$
- $\sigma_{lessEq}(\langle d_1, d_2 \rangle) = \texttt{max}(0, d_1 - d_2)$
- $\sigma_{greater}(\langle d_1, d_2 \rangle) = \texttt{max}(0, d_2 - d_1 + 1)$
- $\sigma_{greaterEq}(\langle d_1, d_2 \rangle) = \texttt{max}(0, d_2 - d_1)$

Intuitivamente, estas funciones de consistencia cuantifican la inconsistencia de una tupla dada por medio de su *diferencia* con respecto a una tupla consistente. Por ejemplo, dada la tupla $\langle 4, 5 \rangle$, la función σ_{less} retorna 0 porque la tupla es claramente consistente. Para una tupla inconsistente, e.g., $\langle 8, 5 \rangle$, la misma función retorna 4, valor que representa la magnitud de la inconsistencia asociada con la tupla.

En Mozart, dichas restricciones pueden imponerse de la siguiente forma:

$$\{\texttt{Soft.relOp D1 D2}\ \beta\},$$

donde `relOp` (que representa los operadores en $RelOp$) debe reemplazarse por `less`, `lessEqual`, `greater` o `greaterEqual`, según sea el caso.

Restricciones Aritméticas para las operaciones básicas (suma, multiplicación, resta, división, módulo y potenciación) representadas por los operadores en el conjunto $ArithOp = \{+, \times, -, \div, \texttt{mod}, \texttt{pow}\}$. Dadas X, Y y Z (tres variables de dominio finito) todas las restricciones están definidas por la siguiente función de consistencia:

$$\sigma_{\texttt{arithOp}}(\langle d_1, d_2, d_3 \rangle) = |d_3 - (d_1\ \texttt{arithOp}\ d_2)|. \qquad (\texttt{arithOp} \in ArithOp)$$

Informalmente, esta función de consistencia considera la diferencia entre una tupla dada y una tupla correcta, sin considerar el "sentido" que tenga la inconsistencia. Esto se justifica si se considera que el objetivo es relajar la igualdad inducida por la restricción fuerte. Por ejemplo, para la restricción de suma, la función $\sigma_{\texttt{plus}}$ asigna el mismo valor

(i.e., 2) para las tuplas $\langle 5, 2, 9\rangle$ y $\langle 5, 2, 5\rangle$, que son igualmente inconsistentes con respecto a la tupla $\langle 5, 2, 7\rangle$.

La sintaxis en Mozart para este tipo de restricciones es la siguiente:

$$\{\texttt{Soft.arithOp D1 D2 D3 } \beta\},$$

donde `arithOp` debe reemplazarse por alguna de las terminaciones siguientes: `plus`, `times`, `minus`, `intDivision`, `modI`, `pow`.

Restricciones Globales sobre variables de dominios finitos. En particular:

- Una versión débil de la restricción de *distancia* para tres variables de dominios finitos. En su versión original [SS04], para un operador $rel \in RelOp$, esta restricción crea un propagador para $|D1 - D2|$ rel $D3$. La función de consistencia asociada se basa en las funciones de consistencia para restricciones relacionales (descritas antes) de la siguiente forma:

  $$\sigma_{\texttt{distance}}(\langle d_1, d_2, d_3\rangle) = \sigma_{rel}(\langle |d_1 - d_2|, d_3\rangle).$$

  La sintaxis en Mozart para esta restricción es:

  $$\{\texttt{Soft.distance D1 D2 rel D3 } \beta\}.$$

- Una versión débil de la restricción *distinct* sobre una lista de variables de dominios finitos. La función de consistencia es igual a la discutida en el capítulo anterior (t es la lista de dominios finitos):

  $$\sigma_{\texttt{distinct}}(t) = nrepeat(t).$$

  La sintaxis en Mozart para esta restricción es:

  $$\{\texttt{Soft.distinct Dv } \beta\}.$$

- Una versión débil de la restricción *distinctOffset*. Dado un vector de números enteros `I` y una lista de variables de dominios finitos `Dv` (ambos de longitud `L`), esta restricción puede describirse como el siguiente ciclo:

```
for K in 1..L-1
  for J in K+1..L
      Dv.K + I.K ≠ Dv.J + I.J
   end
end.
```

Por esta razón, la función de consistencia para esta restricción es igual a la de la restricción *distinct*, pero esta vez sobre la secuencia t^* que representa la suma vectorial de la lista `Dv` y el vector `I`. Formalmente, t_i^*, la i-ésima posición de esta secuencia combinada, es igual a `Dv`.i + `I`.i. Por lo tanto:

$$\sigma_{\texttt{distinctOffset}}(\texttt{Dv}, \texttt{I}) = nrepeat(t^*).$$

Finalmente, la sintaxis de esta restricción en Mozart es:

$$\{\texttt{Soft.distinctOffset Dv I } \beta\}.$$

4.2. Criterios de Distribución

Como se mencionó en el capítulo 2, para garantizar completitud en la solución de problemas es necesario complementar los procesos de propagación con una etapa de distribución. La selección de la variable a distribuir es un proceso critico, pues puede ser la diferencia entre un proceso de búsqueda eficiente y uno computacionalmente costoso [CFMM05].

En el caso de las restricciones clásicas de dominio finito, existen diversos criterios de distribución [SS04] que pueden ser convenientes dependiendo del tipo de problema que se esté solucionando. De la misma forma, criterios de distribución que se ajusten a la filosofía y funcionamiento del módulo de restricciones débiles deben ser desarrollados. En este sentido, los factores de penalización asociados con las restricciones pueden jugar un papel fundamental en la definición de estrategias de distribución para problemas de satisfacción de restricciones débiles, pues proveen información adicional sobre el problema. Al igual que en el caso clásico, el desarrollo de diversos criterios de distribución puede facilitar el modelamiento y solución de diferentes tipos de CSPs débiles.

Creemos que los criterios de distribución para CSPs débiles deben basarse en la premisa de que la variable con mayor número de propagadores asociados debe distribuirse primero. De esta forma, la instanciación de dicha variable tendría efectos inmediatos en el trabajo de filtrado de los propagadores asociados. En nuestro caso, en donde restricciones clásicas y débiles pueden coexistir en el mismo programa de restricciones, la instanciación de las variables considerando el *número total* de propagadores asociados (con la capacidad de distinguir entre propagadores fuertes y débiles) puede resultar fundamental para alcanzar un mayor nivel de poda sobre los dominios de las variables.

Dado que es posible que dicho criterio resulte insuficiente para aquellas situaciones en donde

muchas variables tienen asociados el mismo número de propagadores, proponemos tres criterios de desempate que toman en cuenta los elementos característicos del módulo de restricciones débiles. Todos, de diferentes formas, pretenden encontrar las restricciones débiles *más fuertes* posibles:

Mayor promedio de factores de penalización (*average*) Para todas las variables involucradas en empate, este criterio extrae el *promedio* de los factores de penalización relacionados con las restricciones asociadas con ellas.

Mayor factor de penalización (*highest*) Este criterio selecciona la variable que tenga asociada la restricción con el factor de penalización más alto, basándose en el hecho de que las restricciones con factores de penalización cercanos a 1 tienen menos tolerancia a violaciones, lo que a su vez, puede disminuir el tamaño del espacio de búsqueda.

Mayor número de propagadores débiles asociados (*soft*) La instanciación que tiene lugar por la distribución de una variable puede ser más efectiva si repercute en el mayor número de restricciones débiles. Aquí *efectiva* se relaciona con los procesos de poda que pueden ahorrarse por tal instanciación.

El siguiente ejemplo ilustra la funcionalidad de estos criterios de desempate.

Ejemplo 4.1. *Considere tres variables de dominio finito X_1, X_2 y X_3 y cinco restricciones débiles ($c_1, \ldots, c_5$). Cada una de las variables tiene el mismo número total de propagadores asociados y está relacionada con una o mas restricciones débiles: X_1 está asociada con c_1 y c_2, X_2 con c_1, c_4 y c_5, mientras que X_3 está relacionada con c_3. Los factores de penalización para las restricciones $c_1, \ldots, c_5$ son 0.75, 0.92, 0.90, 0.4 y 0.6, respectivamente.*

- *El criterio de mayor promedio de factores de penalización elegiría la variable X_3 pues su promedio (0.90) es el mayor de las tres variables en cuestión (comparado con 0.58 para X_2 y 0.835 para X_1).*
- *El criterio de mayor factor de penalización elegiría la variable X_1, asociada con la restricción c_2 que tiene 0.92 como factor de penalización.*
- *El criterio de mayor número de propagadores débiles asociados escogería la variable X_2.*

Adicional a estas operaciones, los métodos {`Soft.searchAll P`} y {`Soft.searchOne P`} son extensiones estrictas de los métodos de búsqueda disponibles en Mozart, que retornan la valuación asociada con cada solución encontrada para el problema de restricciones definido en el procedimiento `P`.

Por otra parte, los métodos {`Soft.setCLevel` τ} y {`Soft.solveBy 'FuzzySemiring'`} deben utilizarse para establecer el nivel de corte para el problema y para definir que el método de solución será el módulo de restricciones débiles difusas, respectivamente.

Restricciones débiles en programas Mozart

Un ejemplo muy sencillo de la sintaxis de los elementos del módulo de restricciones débiles se describe a continuación.

Ejemplo 4.2. *Considere dos restricciones aritméticas débiles (i.e., $X+Y = Z$ y $X < Y$), con una estrategia de distribución específica a las restricciones débiles. El objetivo es encontrar todas las soluciones con valuación mejor o igual a 0.8.*

```
{Soft.solveBy 'FuzzySemiring'}
{Soft.setClevel 0.8}                                  % Declaración del nivel de corte
  proc{Test Sol}
      X Y Z in                                        % Declaración de variables
      X::1#6  Y::1#5  Z::3#10                         % Especificación de los dominios
      {Soft.plus X Y Z 0.07}                          % Declaración de restricciones
      {Soft.less X Y 0.05}
      Sol = sol(x:X y:Y z:Z)                          % Definición de la variable solución
      {FD.distribute generic(order:Soft.soft) Sol}    % Estrategia de distribución
  end
L = {Soft.searchAll Test}
```

Figura 4.1: Ejemplo de un programa Mozart que incluye restricciones débiles

El fragmento de código descrito en la figura 4.1 representa la implementación de este problema en Mozart utilizando nuestro módulo de restricciones débiles. El resultado de ejecutar {`Soft.searchAll Test`} *se muestra en la figura 4.2. Cada solución tiene asociada su valuación respectiva.*

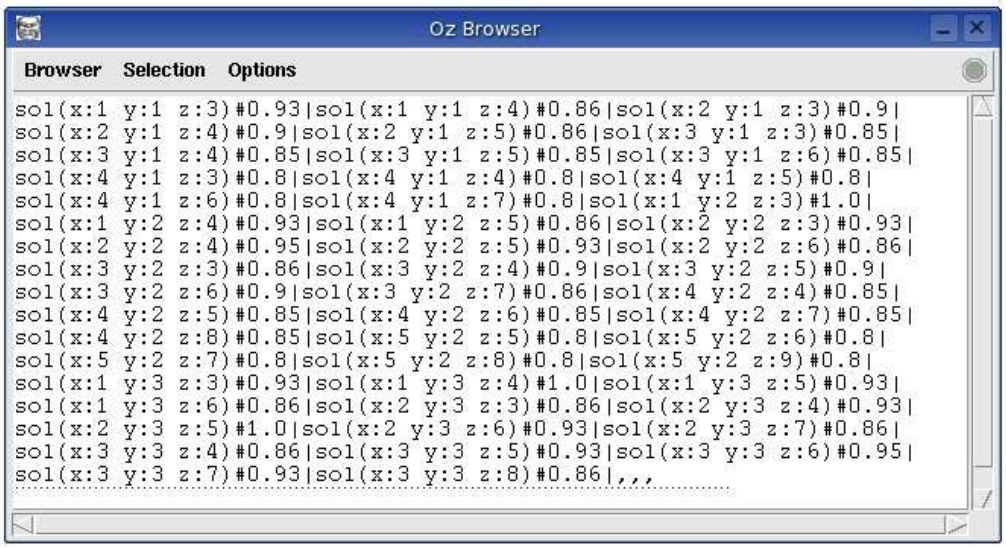

Figura 4.2: Todas las soluciones para un problema de restricciones débiles

4.3. Casos de Estudio

Esta sección se ocupa de ilustrar el uso y aplicación de los dos componentes presentados anteriormente. Los casos de estudio se concentran en resolver instancias particulares de problemas extraídos de la literatura. Inicialmente se presenta la definición completa de cada uno de estos problemas, y luego se procede a describir detalladamente los resultados experimentales obtenidos.

4.3.1. Definiciones

4.3.1.1. Una versión sobre-restringida del problema de n-reinas

Uno de los problemas más conocidos en programación por restricciones es el problema de las n-reinas. Informalmente, el problema consiste en ubicar n reinas en un tablero de ajedrez de n columnas y n filas, de tal forma que las reinas no se ataquen entre si. El modelamiento es muy simple: se asume que las reinas están numeradas de 1 hasta n y que la i-ésima reina se ubica en la i-ésima columna. Por cada reina i, existe una variable R_i que representa la fila en que debe ubicarse. De esta forma, solo resta asegurar que los valores de las variables R_i sean diferentes entre si.

Para hacer sobre-restringido este problema, se asume un tablero de $n \times n - 1$ posiciones. Es

posible modelarlo como se describe arriba, con la condición de que el dominio de las variables debe estar acotado por 1 y $n-1$. De esta forma, la única restricción que debe relajarse para que el problema tenga solución es la que establece que las filas en que las reinas se ubican deben distintas.

4.3.1.2. El Problema de la Fiesta Progresiva (PPP)

El Problema de la Fiesta Progresiva [SBHW95] se sitúa en el contexto de una competencia de yates, en la cual se desea organizar una "fiesta progresiva". Algunos de los yates son designados como *anfitriones*, mientras que las tripulaciones de los yates restantes (*invitados*) visitan los yates anfitriones durante un número fijo de periodos de tiempo consecutivos. El problema consiste en planear estas visitas considerando las siguientes restricciones:

- Una tripulación visitante no puede visitar una tripulación anfitriona dos veces.
- Dos tripulaciones invitadas no pueden encontrarse en un yate anfitrión mas de una vez.
- La capacidad de los yates anfitriones debe respetarse. En la formulación original del problema, además del espacio físico, la capacidad neta de un yate anfitrión esta determinada por las comodidades requeridas por los invitados (por ejemplo, asientos y bebidas) así como por el tamaño de la tripulación anfitriona.

En la formulación del problema real se incluyen condiciones adicionales. Para tener mayor flexibilidad en caso de imprevistos, el yate del organizador de la competencia obligatoriamente debía considerarse como yate anfitrión. Adicionalmente, y considerando que dos yates tenían tripulaciones de padres con niños pequeños, dichas tripulaciones fueron divididas de forma tal que los padres conformaban la tripulación de un yate anfitrión y los niños conformaban un yate virtual con capacidad cero. Un tercer bote virtual fue conformado por los hijos del organizador de la competencia. Los datos originales del problema se describen en el cuadro 4.1.

4.3.2. Resultados Experimentales

La evaluación práctica del módulo de restricciones débiles y de los criterios de desempate se describe a continuación. Todos los resultados presentados son el promedio de cincuenta ejecuciones y fueron obtenidos utilizando un computador con un procesador Pentium Xeon a 2.4 GHz y 1 GB de RAM, usando Mozart 1.3.1.

Yate	Capacidad	Trip.	Yate	Capacidad	Trip.	Yate	Capacidad	Trip.
1	6	2	15	8	3	29	6	2
2	8	2	16	12	6	30	6	4
3	12	2	17	8	2	31	6	2
4	12	2	18	8	2	32	6	2
5	12	4	19	8	4	33	6	2
6	12	4	20	8	2	34	6	2
7	12	4	21	8	4	35	6	2
8	10	1	22	8	5	36	6	2
9	10	2	23	7	4	37	6	4
10	10	2	24	7	4	38	6	5
11	10	2	25	7	2	39	9	7
12	10	3	26	7	2	40	0	2
13	8	4	27	7	4	41	0	3
14	8	2	28	7	5	42	0	4

Cuadro 4.1: Datos de Entrada para el Problema de la Fiesta Progresiva

4.3.2.1. Evaluación de rendimiento del módulo de restricciones débiles

Esta sección se ocupa de caracterizar el rendimiento del módulo de restricciones débiles en el contexto particular del PPP. Los experimentos descritos aquí intentan demostrar la utilidad del uso de restricciones débiles en problemas difíciles y estudiar la influencia de los parámetros del módulo sobre el rendimiento general del sistema.

Inclusión de restricciones débiles en problemas difíciles Para analizar las ventajas del uso de restricciones débiles, se implementó el modelo para el PPP descrito en [SBHW95]. Los datos de entrada considerados fueron los mismos datos descritos en el cuadro 4.1, para 6 periodos de tiempo.

La primera prueba consistió en determinar la dificultad de resolver en Mozart esta instancia del PPP. Los resultados fueron contundentes: después de más de 12 horas de procesamiento continuo, no fue posible encontrar una solución.

Estos resultados sugirieron entonces un debilitamiento del problema. Luego de analizar el conjunto de restricciones y de ponderar los efectos de debilitar una u otra, se decidió relajar el requerimiento de que una tripulación invitada no puede visitar una embarcación anfitriona más de una vez. Esta restricción, que en el modelo fuerte se implementa con una restricción `FD.distinct`, en el modelo débil se implementó con la restricción `Soft.distinct`, con la siguiente condición: sólo se permitió un máximo de dos ocasiones en las que una trip-

Nodos Totales	Nodos Fallidos	Profundidad	Tiempo (s)
56168	56010	1083	76.53407

Cuadro 4.2: Resultado de procesar una versión débil del PPP

ulación invitada puede visitar una anfitriona. Esta relajación controlada se logró eligiendo valores apropiados para el nivel de corte y el factor de penalización asociado con la restricción `Soft.distinct`; en nuestro caso, dichos valores fueron 0.85 y 0.1, respectivamente.

El resultado de procesar esta versión débil del problema original es muy significativo: se encontró una solución (valuada con 0.9) en poco más de un minuto de procesamiento. El cuadro 4.2 describe información más detallada de este proceso de solución.

Este caso demuestra la utilidad de las restricciones débiles en escenarios sobre-restringidos o en problemas donde la duración de los procesos de búsqueda de soluciones es tan prolongada que esperar una respuesta por parte del sistema no es viable. La relajación inducida por las restricciones débiles puede, en algunos casos, acelerar la búsqueda de soluciones y/o contar con soluciones aproximadas aceptables.

Influencia del nivel de corte en el rendimiento del sistema Los resultados del experimento anterior sugieren la existencia de una relación estrecha entre los parámetros de un CSP débil y el rendimiento general del sistema. El objetivo del experimento que se describe a continuación fue tener una idea más concreta de tal relación.

Con estas condiciones, `SoftQueens`, un programa en Mozart que implementa la versión sobre-restringida del problema de las n-reinas fue implementado. Inicialmente, para un $n = 9$, se estableció 0.9 como nivel de corte y 0.1 como nivel de penalización asociado con la restricción `Soft.distinct`. De esta forma aquellas soluciones que tuvieran máximo dos reinas en la misma fila eran aceptadas como soluciones. Bajo estas condiciones, el proceso de búsqueda de *todas* las soluciones[1] dio cuenta de 572 soluciones. Este fue el punto de partida para el análisis del propósito inicial del experimento: determinar la relación entre el nivel de relajación de las restricciones y el número de soluciones obtenidas. Esta relación, que puede explicarse por el crecimiento del espacio de búsqueda inducido por la relajación de una restricción, tiene un efecto directo adverso sobre el rendimiento del sistema. Los datos presentados en el cuadro 4.3 hacen clara esta situación para diferentes valores de nivel de corte. Es fácil

[1] Utilizando {`Soft.searchAll`}

Nivel de Corte	Nodos Explorados	Soluciones	Tiempo (ms)
0.9	3932 (25)	572	338.47853
0.8	16486 (28)	7996	1159.38821
0.7	43773 (31)	34556	3030.83071
0.6	73435 (31)	68722	5241.12597

Cuadro 4.3: Comparación del rendimiento de {`SoftQueens 9`} para diversos niveles de corte.

observar cómo, mientras el nivel de corte disminuye, el tamaño del problema (expresado en el numero de nodos explorados), el tiempo de procesamiento requerido y el número de soluciones aumentan dramáticamente. También es interesante notar cómo, por la definición del nivel de corte, este aumento vertiginoso en el número de soluciones aceptadas implica directamente un deterioro de la calidad de las mismas. Finalmente, este caso nos permite apreciar cómo la relajación inducida por las restricciones reduce el efecto de los propagadores sobre los dominios de las variables, provocando que muchas de las decisiones deban tomarse en el proceso de distribución, lo cual es evidentemente ineficiente.

4.3.2.2. Evaluación de rendimiento de los criterios de distribución débiles

En esta sección comparamos el desempeño de los criterios propuestos para la selección de variables en los procesos de distribución para CSPs débiles. El caso de estudio en esta comparación es SUBCELAR0, una instancia sobre-restringida de RLFAP, un problema de asignación de frecuencias de radio. La descripción completa de este problema (así como de las instancias SUBCELAR) será presentada en el capítulo 5; para los propósitos de la comparación que nos ocupa, resulta suficiente con conocer los datos consignados en el cuadro 4.4.

Cuadro 4.4: Evaluación de los criterios de distribución: Características de SUBCELAR0

Número de Variables	32
Tamaño de los dominios de las variables (promedio)	41
Restricciones Fuertes	16
Restricciones Débiles	223
Densidad de Grafo	0.4697

SUBCELAR0 incluye restricciones fuertes y débiles: las primeras se modelaron utilizando `Soft.distance`, mientras que las segundas fueron especificadas usando `FD.distance`, del módulo de dominios finitos provisto por Mozart. Con el propósito de obtener la noción más clara posible con respecto al comportamiento de los criterios de distribución, el factor de

Criterio	Nodos Totales	Nodos Fallidos	Profundidad	Tiempo (s)
nbSusps	70	55	70	0.12120
ff	101	85	102	0.12524
naive	356068	356053	89	399.83049
`Soft.soft`	59	43	60	0.18586
`Soft.average`	58	42	57	0.18963
`Soft.highest`	148	133	84	0.39896

Cuadro 4.5: Evaluación de los criterios de distribución: cuadro comparativo del procesamiento de SUBCELAR0 de CELAR con penalizaciones aleatorias entre 0.0075 y 0.01

penalización asociado con cada restricción débil es un número aleatorio entre 0.00075 y 0.01. Finalmente, el nivel de corte utilizado para el problema es igual a 0.6.

La comparación involucra los siguientes criterios de distribución: `ff` (*first-fail*), `naive` y `nbSusps` (disponibles en el módulo de dominios finitos de Mozart) y `Soft.average`, `Soft.soft` y `Soft.highest` (i.e., los criterios de desempate descritos anteriormente). Los resultados de la comparación, presentados en el cuadro 4.5, describen información tanto del espacio requerido para el procesamiento (en términos de los nodos del árbol de búsqueda) como del tiempo necesario para encontrar la primera solución al problema.

De los resultados puede apreciarse que, para esta instancia en particular, dos de nuestros criterios de distribución son más eficientes que los criterios de Mozart en consumo de memoria, aunque no en tiempo de ejecución. Esta particularidad puede explicarse por el costo asociado con la identificación de las restricciones débiles y de los factores de penalización asociados con ellas. Otro aspecto crucial a considerar es que en nuestro caso de prueba la mayoría de restricciones son débiles. Esto permitió que nuestros criterios tuvieran mayor relevancia en el proceso de solución. Es posible que para casos en los cuales la proporción de restricciones fuertes y débiles sea distinta, la competitividad de nuestros criterios sea diferente.

Finalmente, la diferencia de comportamiento entre `Soft.soft` y `Soft.average` con respecto a `Soft.highest` puede sugerir patrones de diseño de criterios de distribución. En particular, puede decirse que los criterios de selección de variables que consideran la información relacionada con *todas* las restricciones asociadas a las variables, son más adecuados que aquellos criterios basados en comportamientos o características aisladas. En cualquier caso, pueden existir situaciones en los que dichos criterios "aislados" sean convenientes.

4.4. Discusión

Actualmente, la implementación eficiente de solucionadores de problemas de satisfacción de restricciones débiles puede considerarse un problema abierto. Un factor preponderante en este problema tiene que ver con las propiedades que el operador que describe la combinación de tuplas posee. La idempotencia de este operador es fundamental en el momento de diseñar algoritmos eficientes: si el operador es idempotente (como en el caso de los CSPs difusos) la combinación de tuplas y de valuaciones puede hacerse de forma segura. Si, por el contrario, el operador de combinación no es idempotente, entonces no puede garantizarse que un CSP obtenido como resultado de un algoritmo de consistencia local sea equivalente al CSP de entrada al algoritmo. Otras propiedades deseables, como terminación, también están directamente relacionados con la idempotencia de este operador.

En [CS04] se propone una extensión del algoritmo clásico de arco-consistencia para problemas con operadores no idempotentes. Intuitivamente, se trata de una "compensación" de las valuaciones de las tuplas durante los procesos de poda. El objetivo de esta compensación, que se enmarca en el contexto de los CSPs valuados (VCSPs, Valued Constraint Satisfaction Problems) [SFV95, BFM$^+$96], es el de asegurar la equivalencia entre el CSP original y el CSP resultante del proceso de arco consistencia. Extensiones teóricas y algoritmos son presentados para soportar esta idea. Si bien en [CS04] se presentan resultados interesantes con respecto a la implementación correcta de problemas con operadores no idempotentes, al igual que en el modelo basado en semianillos, en los CSPs valuados se asume un esquema *explícito* de almacenamiento de las valuaciones asociadas con las restricciones. De esta forma, y considerando la equivalencias entre el modelo de VCSPs y el modelo basado en semianillos, implementar las ideas de [CS04] en términos de algoritmos eficientes de consistencia local (en particular, límite consistencia [GSZ05]) con esquemas adecuados de almacenamiento de variables es una opción interesante de trabajo futuro.

4.5. Resumen

En este capítulo hemos presentado una implementación en Mozart de las ideas formales presentadas en el capítulo anterior. Dicha implementación se compone de propagadores que implementan restricciones débiles relacionales, aritméticas y globales, así como de criterios de distribución que tienen en cuenta uno de los elementos característicos del modelo conceptual (i.e., el factor de penalización asociado con cada restricción).

El capítulo presentó evidencia experimental del comportamiento de los componentes propuestos. Dichos resultados experimentales demostraron la utilidad de las restricciones débiles en problemas complejos, por medio de la relajación cuidadosa de ciertas restricciones. En particular, esta estrategia fue utilizada en la solución del problema de la fiesta progresiva, acelerando el proceso de búsqueda de soluciones aproximadas de 12 horas a poco mas de un minuto. Adicionalmente, la relación entre los parámetros del módulo y el rendimiento general del sistema fue estudiada. Nociones claras con respecto a la relación existente entre nivel de corte, espacio de búsqueda del problema relajado, tiempo de ejecución del solucionador y calidad de las soluciones aproximadas obtenidas, surgieron de tal estudio. Finalmente, los criterios de distribución propuestos fueron comparados con los criterios provistos por Mozart. Dicha comparación no sólo revelo un rendimiento competitivo de los criterios en tiempo y espacio, sino que también dio claves para la definición de nuevos criterios de distribución.

Versiones previas de los resultados aquí descritos fueron previamente publicados en [DOPR04, DOPR05a, DOPR05b]. Tanto los criterios de distribución como el caso de estudio de la fiesta progresiva son originales a este trabajo de grado.

5 Un Esquema de Abstracción para CSPs Difusos

Este capítulo presenta una implementación del esquema de abstracción presentado en el capítulo 2 y modificado en el capítulo 3. Dicha implementación se vale de las restricciones sobre dominios finitos de Mozart para solucionar problemas de restricciones débiles expresados en términos de semianillos. Los dos elementos dominantes en el esquema de abstracción, la función de abstracción α y la función de concretización γ, son explícitamente definidos en esta implementación.

Inicialmente, este capítulo presenta las principales características de los elementos que componen la implementación del esquema de abstracción, estableciendo una relación con los elementos descritos en el capítulo 3. El problema de asignación de radiofrecuencias (RLFAP) es utilizado como caso de estudio para analizar el desempeño de los procedimientos implementados. El capítulo finaliza con un análisis comparativo de los resultados teóricos y su aplicación en la práctica.

5.1. Un Esquema de Abstracción para SCSPs en Mozart

Esta sección describe los fundamentos y funcionamiento de la implementación en Mozart del esquema de abstracción propuesto en el capítulo 3.

5.1.1. Función de Abstracción

La implementación en Mozart de la función de abstracción se basa en el concepto de contraparte fuerte discutido en el capítulo 3. Para guardar uniformidad entre las dos alternativas de solución de problemas de restricciones débiles, se implementó una contraparte fuerte por cada una de las restricciones débiles descritas en la sección 4.1. El cuadro 5.1 describe cada una de estas contrapartes.

Restricción Débil	Contraparte(s) Fuerte(s)
`Soft.less` (2)	$X <: Y + H$
`Soft.lessEqual` (2)	$X \leq: Y + H$
`Soft.greater` (2)	$Y <: X + H$
`Soft.greaterEqual` (2)	$Y \leq: X + H$
`Soft.plus / Soft.minus` (3)	$X + Y - Z \leq: H$ $Z - X - Y \leq H$
`Soft.times` (3)	$X \times Y \geq Z - H$ $X \times Y \leq Z + H$
`Soft.intDivision` (3)	$\{\texttt{FD.divI}\ X\ Y\} \geq Z - H$ $\{\texttt{FD.divI}\ X\ Y\} \leq Z + H$
`Soft.modI` (3)	$\{\texttt{FD.modI}\ X\ Y\} \geq Z - H$ $\{\texttt{FD.modI}\ X\ Y\} \leq Z + H$
`Soft.pow` (3)	$\{\texttt{FD.power}\ X\ Y\} \geq Z - H$ $\{\texttt{FD.power}\ X\ Y\} \leq Z + H$
`Soft.distance` (4)	$\{\texttt{FD.distance}\ X\ Y\ Z\ (U - H)\}$ si $Z \in RelOp$ $X - Y \leq: U + H \wedge Y - X \leq: U + H$ si Z es $=$
`Soft.distinct` (1)	$\{\texttt{Repeat X}\} \leq H$
`Soft.distinctOffset` (2)	$\{\texttt{Repeat [X + Y]}\} \leq H$

Cuadro 5.1: Contrapartes Fuertes para las Restricciones Débiles propuestas en el Capítulo 4. {`Repeat X`} denota la operación que retorna el número de elementos repetidos en `X`. [`X + Y`] denota la lista resultante de la suma vectorial entre `X` y `Y` (dos listas de igual longitud).

5.1.2. Función de Concretización

La función de concretización se implementa por medio del procedimiento

{`Soft.getConcreteValuation P Ans`}

que retorna la valuación asociada con `Ans` (una tupla completamente instanciada) en el contexto del procedimiento `P`. Internamente, esta función se implementa invocando al solucionador descrito en el capítulo anterior; esto resulta ser una alternativa razonable en términos de eficiencia, pues como la tupla esta complemente instanciada, los procesos de propagación no se ejecutan, siendo la valuación de la tupla el único proceso a ejecutar.

5.1.3. Algoritmo Iterativo

El algoritmo 5.1 representa el esquema iterativo de abstracción para Mozart. Debe tenerse en cuenta que en algunas características la implementación real puede ser ligeramente diferente.

Algoritmo 5.1 Procedimiento Iterativo para Resolver SCSPs en Mozart

```
IterativeSolving := proc (P, Delta, Inter, Option, BetterCut, OldA, Pess)
 1: if Inter.low == BetterCut then
 2:   Ans = OldA
 3:   Conc = BetterCut
 4: else
 5:   NewSpace = {DefineCut P Inter.low}
 6:   Ans = {Soft.searchOneSpace NewSpace 2 P}
 7:   Conc = {Soft.getConcreteValuation P Ans}
 8: if Ans \= nil then
 9:   OldAns = Ans
10:   if {Or Option=='binary' Option=='Pessimistic'} then
11:     B_local_cut = Inter.low
12:   else
13:     B_local_cut = Conc
14: else
15:   OldAns = OldA
16:   B_local_cut = BetterCut
17: if {ValidarSolucion Delta Inter} then
18:   if Ans \= nil then
19:     Ans#Conc
20:   else
21:     OldAns#B_local_cut
22: else
23:   NewInt = {NewInterval Inter Ans Option Conc Pess}
24:   if {And Ans==nil NewInt.upp - NewInt.low =<0.0 } then
25:     nil
26:   else
27:     {IterativeSolving P Delta NewInt Option B_local_cut OldAns Pess}
```

Los parámetros del algoritmo son un procedimiento `P` (que define un CSP débil), un valor de precisión para el intervalo, un intervalo inicial, una cadena `Option` que determina el modo para seleccionar nuevos intervalos, los mejores valores obtenidos para el nivel de corte y la solución al problema y `Pess`, el valor a usar en la estrategia pesimista de selección de intervalos.

Una ejecución del algoritmo puede explicarse de la siguiente manera. Inicialmente se verifica si el límite inferior del intervalo a evaluar es igual al mejor límite inferior encontrado hasta el momento; esto con el fin de evitar una búsqueda sobre valores del semianillo previamente evaluados. Si este es el caso, se definen los valores de solución y concretización de tal forma que el procedimiento de búsqueda de solución sobre el problema abstracto (la etapa más costosa del esquema) no se ejecute nuevamente y que la función que calcula los nuevos límites del intervalo genere uno nuevo con un límite inferior distinto. En caso contrario (el límite inferior

del intervalo de entrada es diferente al mejor límite inferior), se crea un espacio computacional con `P` y el intervalo Inter (linea 5), se procede a buscar solución en este espacio (linea 6) y a extraer la valuación concreta de la solución encontrada (usando la función γ, en la linea 7).

Seguidamente, se almacenan los valores de la solución y el mejor límite inferior obtenidos hasta el momento, dependiendo del resultado del proceso de solución sobre el problema abstracto y del modo seleccionado de selección de los nuevos intervalos. A continuación, se verifica si el intervalo es lo suficientemente pequeño para ser aceptado por el usuario (i.e., límite superior - límite inferior $\leq$ Delta). En caso de que el tamaño del intervalo sea aceptable, se retorna la mejor solución obtenida, junto con su respectiva valuación en el problema concreto (líneas 18-21). Si es posible seguir acortando el intervalo, se genera uno nuevo, considerando las opciones dadas por el usuario (línea 23). Si el intervalo generado es invalido y no se ha obtenido solución, el procedimiento termina (líneas 24-25). De otro modo, una llamada recursiva considerando los valores obtenidos y el nuevo intervalo generado (NewInt) es realizada (línea 27).

Operaciones Disponibles

La forma de invocar los procedimientos de abstracción desde Mozart es la siguiente:

```
{Soft.abstrae P Delta Start Options}
```

donde:

- `P` es un procedimiento que define un CSP que incluye restricciones débiles.
- `Delta` es un número real entre 0 y 1 que representa la precesión deseada para el tamaño deseado del intervalo de soluciones.
- `Start` es el nivel de corte inicial para el proceso iterativo. Así, el primer intervalo considerado será [`Start`,1.0].
- `Options` representa la estrategia de generación de intervalos deseada. Puede ser `'binary'`, `'eager'` o `'pessimistic'`, según los lineamientos definidos en el capítulo 3.

Procedimientos de Abstracción en programas Mozart

Para utilizar los procedimientos de abstracción en un programa Mozart, la opción {`Soft.solveBy 'FuzzyFD'`} debe ser declarada antes del procedimiento que define el CSP débil. Adicional-

mente, y debido a los procesos de solución que el procedimiento ejecuta, no hay necesidad de invocar métodos de búsqueda de soluciones.

Ejemplo 5.1. *Considere de nuevo el problema descrito en el ejemplo 4.2. El fragmento de código descrito en la figura 5.1 utiliza los procedimientos de abstracción para solucionarlo.*

```
{Soft.solveBy 'FuzzyFD'}
  proc{Test Sol}
      X Y Z in                                          % Declaración de variables
      X::1#6  Y::1#5  Z::3#10                           % Especificación de los dominios
      {Soft.plus X Y Z 0.07}                            % Declaración de restricciones
      {Soft.less X Y 0.05}
      Sol = sol(x:X y:Y z:Z)                            % Definicion de la variable solución
      {FD.distribute generic(order:Soft.soft) Sol}      % Estrategia de distribución
  end
{Soft.abstrae Test 0.001 0.55  'binary'}                % Invocación de la abstracción
```

Figura 5.1: Un programa Mozart que utiliza los procedimientos de abstracción

Es importante resaltar cómo el módulo de restricciones débiles y los procedimientos de abstracción comparten la misma sintaxis para la definición de CSPs con restricciones débiles. Esto se logra definiendo una capa intermedia que, de acuerdo al método de solución elegido, invoca los procedimientos correspondientes.

5.2. Caso de Estudio: El Problema de Asignación de Radiofrecuencias (RFLAP)

En esta sección se ilustran la funcionalidad y principales características del esquema de abstracción implementado. El caso de estudio es una de las instancias del problema de asignación de radiofrecuencias (RLFAP) provista por el CELAR (Centre d' Electronique de l' Armement, Centro Francés de Sistemas Electrónicos de Armamento). Este problema resulta interesante porque permite poner a prueba esta técnica de programación en una situación de la vida real. La instancia particular del problema es sobre-restringida, y puede considerarse compleja en términos del número de variables y de restricciones. Es conveniente mencionar que algunas de las características originales del problema no fueron consideradas en las pruebas que se describen a continuación.

Las resultados descritos aquí fueron obtenidos en las mismas condiciones que los experimentos del capítulo anterior.

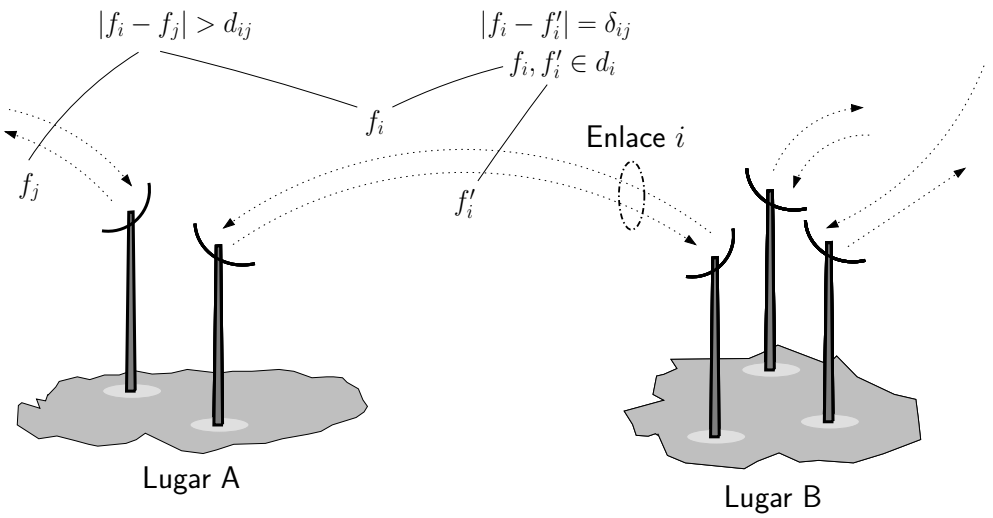

Figura 5.2: Elementos del problema de asignación de radiofrecuencias para dos lugares. Nótese que i es un enlace bidireccional.

5.2.1. Descripción del Problema

El problema de asignación de radiofrecuencias es un problema de dominios finitos que consiste en asignar canales de comunicación a enlaces de radio, usando recursos espectrales limitados. El modelo define una variable por cada enlace de radio; el dominio de dicha variable esta compuesto de las frecuencias disponibles. Varias restricciones, débiles y fuertes, deben satisfacerse en el problema:

- $x_i = f_j$ asigna la frecuencia f_j al enlace x_i. Cuando esta asignación no puede darse, un costo a_i debe pagarse.
- $|x_i - x_j| > d_{ij}$ es usada para modelar la *interferencia* de los enlaces x_i y x_j. Un costo b_i debe asumirse cuando esta restricción no pueda mantenerse.
- $|x_i - x_j| = \delta_{ij}$ define un enlace bidireccional. Se trata de una restricción fuerte que define que la diferencia entre la distancia de la frecuencia asignada a x_i y la frecuencia asignada a x_j debe ser igual a un número fijo δ_{ij}.

La figura 5.2 describe gráficamente la idea detrás del RFLAP.

Nuestro interés se centra en estudiar el comportamiento y desempeño de los procedimientos de abstracción en CELAR 6, la sexta instancia provista por el CELAR. Aunque en la versión original de la instancia se intenta minimizar la suma de los costos de violación, este criterio de

optimización no es considerado en nuestras pruebas. Adicionalmente, y por el tipo de modelo que se asume, el efecto de la violación de múltiples restricciones es diferente al que propone el problema original. De acuerdo con [CdGL$^+$99], durante el proceso de búsqueda de cotas inferiores para CELAR 6, se extrajo un conjunto de subinstancias difíciles de menor tamaño. Por su tamaño y complejidad, dichas instancias (descritas en el cuadro 5.2) resultan apropiadas para medir el desempeño del esquema iterativo de solución basado en la abstracción.

Instancia	No. de Variables	No. de Restricciones	Densidad
6-0	32	223 (16)	0.4697
6-1	28	314 (14)	0.8306
6-2	32	369 (16)	0.7439
6-3	36	439 (18)	0.6968
6-4	44	499 (22)	0.5274

Cuadro 5.2: Instancias extraídas de CELAR 6. El número de restricciones fuertes para cada instancia está denotado con paréntesis.

5.2.2. Comparaciones y Pruebas

5.2.2.1. Comparando el Esquema Abstracto vs. Solucionador Concreto

En esta prueba se comparan los procedimientos de abstracción iterativa y el solucionador de restricciones débiles descrito en el capítulo anterior. El objetivo es encontrar el τ más grande para el problema dado en el menor tiempo posible. Para realizar esta comparación se usa el solucionador de restricciones difusas de la siguiente manera: se supone un valor especifico para τ, se ejecuta el solucionador con tal valor y se verifican los resultados del proceso de búsqueda para elegir el siguiente τ. Es importante resaltar la naturaleza empírica con la que se eligieron los valores de τ para buscar una solución, lo que representa la desinformación que suele haber en este tipo de procesos.

El problema a resolver es la instancia 6-2, asumiendo costos de violación de 0.015 para todas las restricciones débiles del problema. Las restricciones fuertes son modeladas usando las restricciones de dominio finito (FD) de Mozart. Ambos sistemas (el solucionador concreto y los procedimientos de abstracción) consideran inicialmente un $\tau = 0{,}6$. El valor de precisión para el procedimiento de abstracción es igual a 0.01, usando la estrategia ávida para seleccionar el límite inferior de los intervalos generados.

Los resultados de la prueba se muestran en los cuadros 5.4 y 5.3. Para el solucionador concreto de restricciones difusas, cuatro niveles de corte fueron elegidos. Únicamente se encontró solu-

Iteración	Límite Inferior	Límite Superior	Hubo Solución?
1	0.6	1.0	No
2	0.3	0.6	Sí
3	0.45	0.6	No
4	0.31 *	0.45	Sí
5	0.38	0.45	No
6	0.31	0.38	Sí
7	0.345	0.38	Sí
8	0.3625	0.38	No
9	0.355	0.3625	Sí
Tiempo total	8.29221 (s)		

Cuadro 5.3: Descripción del proceso de abstracción iterativa para la instancia 6-2. Los límites inferiores tienen un '*'cuando la valuación de la solución encontrada en el dominio concreto fue mejor que el límite inferior dado por la estrategia binaria.

τ Inicial	Tiempo (s)	Valuación
0.6	5.49996	`nil`
0.5	330.46251	`nil`
0.4	727.95230	`nil`
0.3	327.01492	0.31
Tiempo total	1390.9297 s.	

Cuadro 5.4: Búsqueda de un buen nivel de corte usando un solucionador (concreto) difuso para la instancia 6-2.

ción con el último. Todo el proceso tomó alrededor de 23 minutos.

En contraste, los resultados obtenidos con el procedimiento de abstracción son distintos. Además de las significativas mejoras en tiempo, usando el procedimiento de abstracción es posible encontrar una cota inferior para una solución óptima. En este caso particular, dicha cota es superior al nivel de corte obtenido usando el solucionador concreto, y sólo requirió de nueve iteraciones para alcanzar un intervalo preciso. Un aspecto interesante de esta prueba lo constituye el hecho de que solamente en una iteración (la cuarta) el valor de la mejor solución encontrada hasta el momento fue mejor que el nivel de corte dado por la estrategia binaria.

Es importante analizar las diferencias notorias en rendimiento entre ambos sistemas. En este punto, debe tenerse en cuenta que todas las restricciones débiles del problema son del mismo tipo, lo que nos lleva a pensar que los efectos una implementación poco eficiente de `Soft.distance` son mucho más evidentes que si el problema incluyera restricciones débiles de diversos tipos. A su vez, esta observación permite pensar que las funciones de consistencia y valuación asociadas con la implementación propuesta de `Soft.distance` son susceptibles a

Inst.	Mejor Intervalo Encontrado	Costo de Violación	Nivel de Corte Inicial			
			0.3	0.5	0.7	0.9
6-0	[0.58, 0.592187]	0.02	386.60408 (10)	425.82416 (11)	388.66321 (9)	490.09802 (12)
	[0.685, 0.701875]	0.015	372.29204 (11)	251.14701 (8)	276.75949 (7)	485.84673 (12)
	[0.79, 0.802188]	0.01	344.97944 (9)	365.46499 (10)	358.01650 (9)	451.52191 (11)
6-4	[0.14, 0.152813]	0.02	87928.27452 (5)	274609.04763 (10)	249075.20058 (8)	250126.75751 (9)
	[0.355, 0.374688]	0.015	190345.69032 (11)	131577.26468 (7)	301857.13019 (13)	162980.82373 (9)
	[0.57, 0.582812]	0.01	158341.57401 (10)	161384.60727 (11)	89023.04733 (9)	170547.05327 (12)

Cuadro 5.5: Influencia del nivel de corte inicial en el desempeño de los procedimientos de abstracción. Cada celda contiene el tiempo de ejecución (en milisegundos) y el número de iteraciones requeridas para encontrar el mejor nivel de corte. El valor de precisión utilizado para todas las ejecuciones fue 0.01.

mejoras. Esto también realza el nivel de sofisticación de los propagadores de dominio finito en la implementación actual de Mozart (altamente acoplados con el núcleo del lenguaje) con respecto a implementaciones basadas en los mecanismos de extensión de éste.

5.2.2.2. Elección de Niveles de Corte Iniciales

Esta prueba se encarga de comparar el desempeño de los procedimientos de abstracción para las instancias 6-0 y 6-4. Esta selección de instancias se basa en una de las observaciones descritas en [CdGL$^+$99]: con excepción de 6-0, toda instancia 6-i es una subinstancia de 6-$i+1$ y por lo tanto, presumiblemente más sencilla de resolver. Esto quiere decir que de las subinstancias de CELAR 6, 6-0 y 6-4 son las únicas disjuntas y que, adicionalmente, 6-4 puede considerarse como la más difícil de todas.

Esta prueba tiene varios objetivos. Primero, una inquietud válida con respecto a los procedimientos de abstracción implementados es determinar hasta qué punto un nivel de corte inicial tiene influencia en el desempeño completo del sistema. Para esto se asumen diversos valores para τ, que van de 0.3 a 0.9. Esta influencia es expresada en el número de iteraciones necesarias para alcanzar el intervalo. Segundo, una inquietud con respecto al problema de asignación de radiofrecuencias es la influencia de los costos de violación en el desempeño del sistema. Para esto se consideran tres costos de violación: 0.01, 0.015 y 0.02. Finalmente, es deseable obtener una medida concreta que represente el desempeño en tiempo del sistema. En particular, resulta interesante estudiar el comportamiento del tiempo de ejecución cuando los costos de violación y los niveles de corte iniciales varían como se describieron arriba.

Los resultados de esta prueba están descritos en el cuadro 5.5. La segunda columna representa un *intervalo de referencia*, obtenido de la ejecución de cada una de las instancias utilizando

un nivel de corte inicial de 0.1. De este intervalo es posible apreciar el efecto de los costos de violación sobre los límites del intervalo. Es conveniente mencionar que el intervalo exacto que se obtiene utilizando los diferentes niveles de corte es ligeramente diferente a este intervalo de referencia.

Los resultados permiten extraer varias conclusiones. Inicialmente, se puede apreciar que la instancia 6-0 es significativamente más sencilla de resolver que la instancia 6-4, tal y como se conjetura en [CdGL$^+$99]. Con respecto al primer objetivo del experimento, usando el intervalo de referencia es posible deducir que hacer estimaciones *a la baja* del nivel de corte inicial es una estrategia adecuada para usar el procedimiento de abstracción. En los resultados, este hecho es evidente en la instancia 6-0 que utiliza 0.015 como costo de violación. Aunque en la práctica el nivel de corte óptimo es desconocido, este tipo de estimación puede considerarse como acertado. Con respecto a la relación entre el costo de violación y el número de iteraciones, solamente en una ejecución (instancia 6-4 con nivel de corte de 0.9) se aprecia claramente que el número de iteraciones aumenta a medida que el costo de violación disminuye. Por esta razón, no es posible generalizar el comportamiento del procedimiento de abstracción bajo estos parámetros. Un análisis similar aplica para la relación entre el valor asignado a los costos de violación y el tiempo de ejecución.

En general, puede decirse que el experimento demuestra cuán difícil es extraer observaciones concluyentes a partir de una serie de instancias particulares. Esto puede explicarse por las características específicas de cada problema: por las variaciones dispuestas entre los diferentes parámetros, los valores de holgura resultantes configuran un problema de restricciones diferente para cada caso, así se trate de la misma instancia de problemas. A pesar de lo anterior, es preciso destacar como el comportamiento en términos de ejecución del procedimiento de abstracción es bastante competitivo.

5.3. Contraste Teoría vs. Práctica

En esta sección se evalúa la relación entre el esquema teórico de abstracción para restricciones débiles basadas en semianillos y el modelo iterativo de solución que fue propuesto conceptualmente en el capítulo 3 y descrito en las secciones anteriores.

El principal factor que debe considerarse para realizar esta comparación (o "evaluación de aplicabilidad", como también podría denominarse) tiene que ver con la facilidad de implementación de los resultados teóricos. Más precisamente, puede hablarse de la *naturaleza* de

las premisas teóricas que soportan las propiedades del esquema de abstracción. En ese sentido, la principal premisa es el principio conceptual bajo el cual la estructura principal del problema (variables, dominios, restricciones) permanece intacta luego del proceso de abstracción, mientras que lo que se aproxima son las características del conjunto del cual se toman las valuaciones (es decir, A). En el caso particular de la abstracción implementada, el objeto de la abstracción es el número de elementos de A: un número infinito de posibles valuaciones (los números reales) se aproxima con dos únicos valores (los enteros 0 y 1).

Estrechamente relacionada con esta premisa se encuentra un principio similar que fue identificado, discutido y adaptado en el capítulo 3, y se refiere a que en la manipulación de un problema de restricciones débiles se cuenta con una asociación explícita entre valuaciones y tuplas. Una característica fundamental del modelo de abstracción es que refuerza y extiende este principio, al asumir que por dicha asociación explicita la modificación de las valuaciones debería ser simple e inmediata. En otras palabras, se está asumiendo que los procesos de abstracción y concretización se reducen a un mero reemplazo de las valuaciones asociadas con las tuplas.

Durante el desarrollo de este trabajo la dificultad de conciliar este principio con el objetivo de lograr una implementación eficiente fue un factor preponderante. Las estructuras y procesos necesarios para la implantación del esquema de asociación explícito, nos permiten pensar que los costos asociados con el reemplazo exhaustivo de valuaciones probablemente hubiesen minimizado u ocultado los posibles ahorros derivados del uso de la abstracción. Además, tal fenómeno hubiese sido especialmente notorio en problemas de tamaño mediano/grande, en los cuales el número de variables y de valores por dominios puede ser significativamente grande. De esta forma, y tal y como los resultados experimentales (descritos en la sección anterior) muestran, un reemplazo implícito de valuaciones puede considerarse como una estrategia adecuada en términos de rendimiento y eficiencia.

Por otra parte, el modelo teórico de la abstracción involucra otras ideas, que por su concepción, fueron fácilmente interpretables en el contexto de una implementación. En particular, el esquema iterativo de solución y las propiedades que proveen información segura con respecto a la ubicación de soluciones óptimas resultaron ser de inmensa utilidad en este trabajo. La generalidad asociada con ambos resultados permitió que su implementación en Mozart fuera sencilla. Este proceso de *interpretación* de los resultados teóricos en aplicaciones prácticas, unido con los resultados experimentales ya discutidos, permiten pensar en los procedimientos de abstracción implementados como una alternativa atractiva para la solución de problemas

de restricciones débiles, en donde la extensibilidad y la solidez teórica son sus características fundamentales.

5.4. Discusión

Al igual que la implementación del módulo de restricciones débiles presentado en el capítulo 4, el desarrollo del esquema de abstracción iterativo está marcado por la toma de decisiones de diseño que, en apariencia, no se ajustan totalmente al espíritu de los resultados teóricos que implementan. En particular, es importante referirse al carácter específico que tienen las implementaciones, en contraste al carácter genérico propio de los resultados formales.

En la implementación de este tipo de resultados, debe ponderarse entre la conveniencia de implementar todos los resultados de forma absolutamente genérica o concentrarse en implementar aquellos casos particulares de mayor interés. Como es natural, la primera opción es la favorita entre científicos e investigadores, pues les permite demostrar la pertinencia de sus propuestas. Sin embargo, durante el desarrollo de este trabajo se comprobó la conveniencia de implementar casos particulares de la teoría. En nuestro caso, esta aproximación se justifica en las condiciones asociadas con los resultados principales (como la preservación del orden entre las funciones de abstracción y concretización, o la idempotencia del operador multiplicativo en el procesamiento de SCSPs débiles) que limitan fuertemente su implementación.

Es conveniente destacar algunas de las ventajas de nuestra decisión. Por una parte, una adecuada particularización de los resultados teóricos al momento de su implementación facilita el análisis de la aplicabilidad de las propiedades teóricas en la práctica, pues evita concentrarse en detalles que aportan poco a tal análisis, como las dificultades técnicas surgidas de la implementación de modelos demasiado generales. Vale recordar que dicho análisis es fundamental para los propósitos de este trabajo. Por otra parte, la implementación de casos específicos puede redundar en una mayor eficiencia de las herramientas de software resultantes.

5.5. Resumen

En este capítulo hemos presentado una implementación para Mozart del esquema iterativo de abstracción propuesto en el capítulo 3. Dicha implementación está basada en la manipulación de espacios de computación en Mozart y sus componentes principales son un conjunto de operaciones que implementan las contrapartes fuertes. El esquema iterativo presentado es

completamente extensible, no sólo en cuanto al número y capacidades de las contrapartes fuertes sino también en cuanto a las opciones del algoritmo iterativo en sí. Cabe destacar que el hecho de que el núcleo del esquema está definido en Mozart contribuye grandemente a esta extensibilidad, un aspecto particularmente beneficioso para aquellos usuarios no familiarizados con un lenguaje de más bajo nivel como C/C++.

Adicionalmente, en este capítulo hemos presentado evidencia del comportamiento práctico del esquema iterativo de solución. Dos pruebas principales fueron ejecutadas: la primera se concentró en la comparación del esquema iterativo con respecto a las restricciones débiles presentadas en el capítulo 4, mientras que la segunda procuró la caracterización de la influencia de los parámetros iniciales en el comportamiento general de los procedimientos de abstracción. De los resultados de ambas pruebas fue posible extraer conclusiones positivas con respecto al rendimiento de los componentes implementados.

Los resultados principales de este capítulo fueron publicados previamente en [DPR05]. La presentación de este capítulo incluye nuevos y más detallados datos estadísticos.

6 Análisis y Comparaciones

Este capítulo se encarga de contextualizar la aplicabilidad de los dos componentes prácticos presentados en los capítulos anteriores. Inicialmente, se propone una categorización de los problemas de satisfacción por restricciones. El criterio de categorización es la *debilidad* presente en los problemas. El propósito de esta categorización es servir de base para la identificación precisa de los problemas que son susceptibles de ser modelados y solucionados usando restricciones débiles.

En segundo lugar, este capítulo compara los componentes implementados en términos de tiempo de ejecución y condiciones de aplicabilidad. Consecuentemente, tanto los resultados experimentales presentados en los capítulos 4 y 5 como la categorización mencionada antes constituyen la base para esta comparación. Seguidamente, los mecanismos implementados son comparados con el método de reificación de restricciones, una forma de *satisfacción parcial* provista de forma explicita por Mozart para la solución de problemas sobre-restringidos. Una discusión sobre la pertinencia y aplicabilidad de los métodos de restricciones débiles y de satisfacción parcial acompaña esta segunda comparación.

Finalmente, y con base en las comparaciones y resultados presentados, el capítulo concluye con una discusión sobre la inclusión de restricciones débiles en aplicaciones existentes, resaltando los aspectos más relevantes en tal proceso.

6.1. Una Categorización de Problemas de Satisfacción de Restricciones

A continuación se propone una categorización de problemas de satisfacción de restricciones basada en la debilidad que éstos incluyen.

1. Problemas *totalmente fuertes*, soportados por un modelo compuesto únicamente de re-

stricciones fuertes que funciona bien en un rango relativamente amplio de casos. Es decir, retorna soluciones aceptables en un tiempo razonable para el usuario. En la práctica, este tipo de problemas son escasos y pueden considerarse como el caso ideal en programación por restricciones. Un ejemplo de este tipo de problemas es el problema de asignación de horarios descrito en [FA03] que, usando un conjunto muy simple de restricciones, retorna soluciones adecuadas para el problema real que modela.

2. Problemas *fuertes relajados*, que se ajustan a un modelo en que la mayoría de restricciones son fuertes y algunas pocas son débiles. Son generalmente sobre-restringidos y las restricciones débiles que estos problemas poseen han surgido de un proceso en el que el usuario modifica el modelo original del problema. El objetivo de estas modificaciones es, por lo general, disminuir los recursos usados (e.g., tiempo de ejecución, espacio en memoria) para obtener soluciones. En algunos casos, la relajación al problema original se manifiesta por la modificación de los datos de entrada originales, en lugar de incluir preferencias o restricciones débiles propiamente. Por ejemplo, en el problema de asignación de horarios y salones descrito en [DPP$^+$05], se implementa un esquema de *satisfacción parcial*, *removiendo* restricciones conflictivas y/o modificando los dominios de las variables, previo consenso con el usuario final.

   En otros casos, la relajación del problema consiste en el *reemplazo* de restricciones fuertes con restricciones débiles. Un ejemplo de este tipo de relajación fue discutido en el capítulo 4 en la solución del PPP: una modificación estratégica permitió acelerar significativamente el proceso de encontrar una solución aceptable.

3. Problemas *débiles fortalecidos*, representados por modelos en los que la mayoría de las restricciones son débiles y unas pocas condiciones esenciales son establecidas como fuertes. Estas restricciones débiles se han definido así por "convicción" y no son resultado de modificaciones de modelos previos del problema. Un ejemplo cercano de este tipo de problemas es RLFAP, discutido de forma detallada en el capítulo 5.

4. Problemas *totalmente débiles*, cuyos modelos carecen de condiciones obligatorias, por lo que todas las restricciones pueden considerarse como débiles. Algunos de los problemas que se ubican en esta categoría son aquellos que se concentran en satisfacer un conjunto de preferencias o deseos de un usuario. Ejemplos de esta situación son los problemas que pueden modelarse con las CP-Networks [RS98], un formalismo que, dadas una serie de condiciones deseables, retorna una solución que intenta satisfacerlas de la mejor manera.

6.2. Comparaciones

En esta sección se comparan los dos componentes desarrollados en este trabajo, el módulo de restricciones débiles y el esquema iterativo de abstracción. Inicialmente se analizan sus alcances y particularidades, y en base a la categorización presentada arriba, se precisan los contextos más convenientes para su aplicación. Más adelante, el módulo de restricciones débiles es contrastado con el esquema de *reificación* de restricciones, el medio por el cual se suelen resolver problemas sobre-restringidos en Mozart.

6.2.1. Restricciones Débiles y Esquema Iterativo de Abstracción

A continuación comparamos las contribuciones prácticas de este trabajo de grado bajo dos criterios principales: rendimiento (tiempo de ejecución) y contextos de aplicabilidad.

Las diferencias en tiempo de ejecución entre los componentes de software propuestos fue analizada en un caso particular en el capítulo anterior. En dicho experimento, el comportamiento en tiempo de ejecución de los procedimientos de abstracción fue significativamente superior al del módulo de restricciones débiles. La conclusión principal del experimento atribuyó este comportamiento a la diferencias de implementación existente entre las restricciones provistas por Mozart (base de las contrapartes fuertes) y las disponibles en el módulo de restricciones débiles.

En términos generales, puede decirse que la diferencia entre el rendimiento de ambos sistemas se reduce al comportamiento de las implementaciones de las contrapartes fuertes; los demás procesos en el esquema de abstracción (e.g., la generación de nuevos intervalos) pueden considerarse como irrelevantes en términos de rendimiento. De esta forma, para alcanzar un rendimiento comparable en ambos sistemas es necesario estudiar cuidadosamente la definición de contrapartes fuertes que sean tanto eficientes como expresivas (en términos de la semántica débil asociada con cada restricción fuerte).

Con respecto a la aplicabilidad de los componentes implementados, con base en la categorización dada anteriormente, y considerando los resultados experimentales discutidos en los capítulos 4 y 5, se pueden extraer varias conclusiones. Inicialmente, es posible afirmar que abordar los problemas de todas las categorías utilizando procedimientos que implementan restricciones débiles puede resultar inconveniente por diferentes razones. En el caso de los problemas *totalmente fuertes*, no es adecuado simular las restricciones fuertes usando restric-

ciones débiles, por los costos que esto acarrea. Como se justificará más adelante, un aspecto crucial en el modelamiento de problemas de la vida real es el uso racional de las restricciones débiles. Por otra parte, el modelamiento y solución de problemas *totalmente débiles* usando alguno de los mecanismos de restricciones débiles propuestos puede resultar extremadamente costoso. Para este tipo de problemas, una estrategia prudente consiste en "fortalecer" paulatinamente el problema original hasta donde sea posible, reconsiderando las características de algunas de las restricciones originales. De esta forma, se puede intuir un escenario de aplicación bien definido para los dos componentes implementados: el módulo de restricciones débiles debe concentrarse en la solución de problemas *fuertes relajados*, mientras el esquema iterativo de abstracción se ajusta mejor a las características de los problemas *débiles fortalecidos*.

En primer lugar, resolver problemas fuertes relajados utilizando el módulo de propagadores débiles es conveniente por las pocas restricciones débiles que debería contener un problema en dicha categoría. De esta forma, es posible obtener soluciones aproximadas para el problema, con un incremento en recursos que puede ser controlado directamente. Tal control se facilita si se tiene en cuenta que, por la poca cantidad de restricciones débiles, en un problema fuerte relajado se tienen indicios serios sobre la satisfacción global del problema y que la definición de los elementos parametrizables del módulo está orientada hacia lograr dicho control. Finalmente, debe considerarse que es posible lograr ahorros en recursos si los lineamientos de relajación son adecuadamente determinados. Este aspecto será estudiado más detalladamente en la sección 6.3.

En segunda medida, el contexto de un problema débil fortalecido es generalmente diferente al de un problema fuerte relajado. Frecuentemente, por la cantidad de restricciones débiles, en este tipo de problemas es difícil tener una idea de la satisfacción global del problema. El uso de la abstracción en estos problemas resulta conveniente si se considera que el procesamiento del enorme espacio de búsqueda puede facilitarse por las aproximaciones (i.e., ahorros) relacionadas con dicha técnica. Muy posiblemente, el módulo de restricciones débiles fallaría en dar una respuesta aceptable en un tiempo razonable para un espacio de búsqueda tan grande. Cabe anotar que en este caso también se cuenta con elementos de control que pueden guiar y/o facilitar el procesamiento del problema en cuestión. Adicionalmente, por el uso del esquema iterativo de abstracción se pueden obtener indicios concretos (expresados en la evolución del intervalo de soluciones óptimas) sobre la dificultad real del problema.

La figura 6.1 ilustra estas ideas en el contexto de la categorización de problemas propuesta anteriormente.

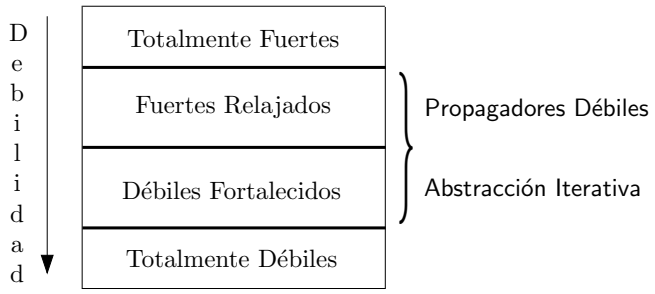

Figura 6.1: Una Categorización de Problemas de Satisfacción de Restricciones

6.2.2. Restricciones Débiles y Restricciones Reificadas

La forma más común de resolver problemas sobre-restringidos en Mozart es por medio de *restricciones reificadas* [SS04]. Informalmente, la reificación de una restricción relaciona su satisfacción con una variable booleana. Por lo general, se reifican varias restricciones y se optimiza alguna función objetivo que incluya las variables booleanas respectivas. Formalmente, la reificación de la restricción c con respecto a una variable x es la restricción $(c \leftrightarrow x = 1) \wedge x \in 0\#1$. Esta nueva restricción está definida por las siguientes reglas de propagación: si la restricción $x = 1$ (resp. $x = 0$) se deduce del almacén, entonces el propagador reificado reduce a un propagador para c (resp. $\neg c$); si del almacén se deduce c (resp. si el almacén es inconsistente con c) entonces el propagador reificado impone $x = 1$ (resp. $x = 0$). Usando este esquema, dado un conjunto I de restricciones reificadas, los usuarios pueden ponderar la importancia de una restricción $c_i \in I$ en el problema completo por medio de un coeficiente a_i, y calcular un valor global $Sat = \sum a_i \times x_i$ de satisfacción para el problema. El valor de Sat es un indicador de calidad para las soluciones, y por lo tanto, un criterio de decisión que permite descartarlas o aceptarlas.

Las restricciones reificadas ofrecen varias similitudes y diferencias con respecto a las restricciones débiles desarrolladas en este trabajo. Por un lado, ambos esquemas ofrecen medios tanto para expresar la importancia relativa de cada restricción en el problema completo como para definir criterios de selección de soluciones: el factor de penalización y el nivel de corte en el módulo de restricciones débiles y los coeficientes a_i y el criterio *particular* de selección de soluciones derivado del valor global de satisfacción del problema (i.e., Sat) en las restricciones reificadas.

Por otro lado, una de las diferencias entre ambos esquemas tiene que ver con la *generalidad* de los criterios de selección de soluciones: mientras que en el módulo de restricciones

débiles tanto el rol como el significado del nivel de corte están formalmente definidos, en el esquema de reificación el criterio de selección varía de problema a problema, lo que impide su generalización. Otra diferencia importante tiene que ver con el *tipo de relajación* que cada mecanismo pretende para el problema. Mientras que en los formalismos de restricciones débiles la debilidad se entiende como una relajación en los *efectos* de las restricciones sobre los dominios de las variables, considerando *todas* las restricciones del problema, en el esquema de restricciones reificadas la relajación está basada en la satisfacción de sólo un *subconjunto* de las restricciones definidas en el problema. Consideramos que esta diferencia conceptual resulta fundamental en este trabajo, en cuanto impide una comparación objetiva entre ambos esquemas de satisfacción.

Estas diferencias sugieren un debate más detallado sobre la aplicabilidad de los modelos e implementaciones de restricciones débiles y de los mecanismos de *satisfacción parcial* de restricciones, idea general en la que se encuentran los métodos de reificación. En primer lugar, debe decirse que por las características discutidas antes, las técnicas de satisfacción parcial son difícilmente generalizables a un amplio espectro de casos, prolongando así una idea largamente arraigada con respecto a la naturaleza *artesanal* de la programación por restricciones. En este sentido, es posible afirmar que, de alguna manera, la generalidad de los modelos de restricciones débiles contribuye en parte a deslegitimar dichas ideas preconcebidas que limitan la popularización de las técnicas de programación por restricciones.

En segundo lugar, debe considerarse que el hecho de que todas las restricciones sean consideradas en la búsqueda de soluciones aproximadas tiene un efecto directo adverso en el rendimiento de dicha búsqueda. Esto es claro si se considera que debilitar los criterios de consistencia de restricciones —base de la idea subyacente a las restricciones débiles— resulta mucho más costoso que ignorar algunas restricciones, como hacen los modelos de satisfacción parcial de restricciones como Max-CSP [Fre89]. De esta forma, en ciertas circunstancias (como en instancias de problemas fuertes relajados) el uso de técnicas de satisfacción parcial puede resultar más conveniente, en términos de rendimiento, que la aplicación de técnicas de restricciones débiles. En cualquier caso, sin embargo, debe considerarse que bajo las ideas de nuestra implementación, incluir una restricción (débil) es más conveniente (en términos de eficiencia) que removerla, pues de esa forma se garantiza algún grado de poda sobre los dominios de las variables.

En este sentido, cabe anotar que este efecto del debilitamiento sobre la eficiencia de la búsqueda de soluciones enfatiza la necesidad de formas explicitas de *controlar* la relajación de los

criterios de consistencia. Es decir, debe ser posible gozar de los beneficios de considerar todas las restricciones de un problema en la búsqueda de soluciones aproximadas, con niveles aceptables de rendimiento. Las características de los componentes implementados (en particular, las opciones de parametrización por parte del usuario) permiten lidiar, de manera directa, con los efectos en rendimiento de la relajación de los problemas.

6.3. Inclusión de Restricciones Débiles en Aplicaciones Existentes

Un buen número de problemas combinatorios poseen características y condiciones que los hacen sobre-restringidos. Para solucionarlos es necesario inducir algún tipo de relajación, transformando el problema de totalmente fuerte a fuerte relajado. En esta sección se discuten las consideraciones que deben tenerse en cuenta para incluir restricciones débiles en problemas que han sido formulados inicialmente como problemas totalmente fuertes. Dicha inclusión es un asunto delicado que gira alrededor de dos factores principales:

- Las modificaciones necesarias en aquellas aplicaciones existentes que desean incluir restricciones débiles;
- Las consecuencias de obtener soluciones usando un modelo que incluye restricciones débiles.

El primer aspecto está relacionado con el costo de introducir restricciones débiles en aplicaciones existentes. Si bien las restricciones débiles permiten describir modelos más precisos, introducir (o convertir) todas o la mayoría de restricciones en el modelo como restricciones débiles es computacionalmente costoso, particularmente por el reducido poder de poda asociado con los propagadores débiles (con respecto a sus análogas fuertes). Adicionalmente, los costos asociados (en tiempo y en dinero) con el cambio de los modelos subyacentes a una aplicación existente basada en restricciones son enormes. La conversión exitosa de un problema totalmente fuerte sobre-restringido a un problema fuerte relajado depende ampliamente de la identificación de un conjunto reducido de restricciones a ser relajadas. Es altamente deseable que dichas restricciones reflejen las características opcionales del problema. Por ejemplo, en los problemas de investigación de operaciones algunas de las restricciones relacionadas con el número de recursos disponibles pueden ser relajadas, dado que algún tipo de concesión o arreglo es posible en la vida real. Por el contrario, aquellas restricciones que establecen las condiciones obligatorias del problema deberían conservarse como restricciones fuertes. Clara-

mente, tanto la escogencia de las restricciones a relajar como el tipo de la relajación debe considerarse cuidadosamente; al final del proceso, los resultados de dichas decisiones condicionaran la calidad y/o utilidad de dichas soluciones.

Un ejemplo de la aplicación de esta estrategia fue descrito en el capítulo 4 en el momento de resolver el problema de la fiesta progresiva. Allí, debido a la dificultad de encontrar una solución en un tiempo prudencial, se analizaron las restricciones definidas en el problema con el fin de seleccionar la más adecuada para ser debilitada. Criterios de conveniencia particulares al contexto de la fiesta progresiva fueron tenidos en cuenta: por ejemplo, tener varios encuentros de tripulaciones invitadas en una misma embarcación puede ser desastroso para la fiesta en general e iría en detrimento de los recursos asignados a otras tripulaciones invitadas. Así, una vez seleccionada la restricción a debilitar, se definieron valores para el factor de penalización y el nivel de corte de tal forma que por su combinación se relajara lo menos posible el problema, con la idea de ir aumentando paulatinamente el nivel de relajación hasta encontrar una solución en un tiempo razonable.

El segundo aspecto concierne al *manejo* de una solución proveniente de un modelo relajado. Aunque contar con una solución para un problema que antes no tenía es de por si un avance significativo, por lo general el uso de modelos de restricciones débiles involucra una "negociación" entre los expertos en restricciones y el usuario final. En dicha negociación el usuario procurará obtener soluciones que cumplan tantas condiciones del modelo real como sea posible. Hay que tener en cuenta que dichas soluciones aproximadas requerirán de esfuerzos por parte del usuario final en el contexto específico del problema. Es en este punto en donde se hace crucial una elección adecuada de restricciones fuertes a relajar, pues este proceso de negociación entre usuario final y desarrollador puede verse beneficiado por la escogencia adecuada de dichas restricciones. De esta forma, una escogencia poco cuidadosa de los elementos del problema a relajar puede resultar inconveniente tanto para el usuario final como para el experto en restricciones.

Retomando el ejemplo del problema de la fiesta progresiva, puede apreciarse como la relajación efectuada sobre el problema afectó en una medida muy pequeña el problema original. Por un lado, la selección de la restricción relajada puede considerarse adecuada si se tiene en cuenta que por su relajación no se atentó contra las especificaciones físicas del problema, lo que posibilita su aplicación y/o adaptación en el contexto real del problema. Por otra parte, la solución aproximada obtenida de esta relajación dista ligeramente de una solución al problema original; esto se debe a que la conjunción entre el nivel de penalización de la restricción y el

nivel de corte del problema sólo permite una violación en la restricción seleccionada, lo que al final redunda en un rendimiento aceptable para aproximar una solución.

En resumen, la inclusión de restricciones débiles en problemas existentes puede ser extremadamente beneficiosa para una gran cantidad de problemas, pero su uso debe ser planeado cuidadosamente. La tarea principal de este proceso, que debe llevarse a cabo conjuntamente entre el experto en restricciones y el usuario final, es seleccionar y relajar adecuadamente ciertos aspectos fundamentales en el problema. Esta selección condicionara aspectos posteriores del proceso de solución, pues las soluciones aproximadas (obtenidas de un problema relajado) pueden ser adoptadas de forma sencilla por el usuario final si los cambios, arreglos y/o concesiones que él debe realizar son razonables en el contexto particular del problema.

6.4. Resumen

En este capítulo hemos presentado análisis y comparaciones de las contribuciones prácticas de este trabajo de grado. La base de estos estudios fue una *categorización* de problemas de satisfacción de restricciones, que caracteriza cuatro tipos de CSPs según la debilidad incluida en ellos. Se discutió cómo estos tipos de problemas sugieren el uso de mecanismos de solución específicos. Por ejemplo, aquellos problemas en donde la mayoría de restricciones son fuertes son susceptibles de ser resueltos usando las restricciones débiles presentadas en el capítulo 4, mientras que los problemas en donde la mayoría de restricciones son débiles se prestan para ser resueltos usando los procedimientos iterativos de abstracción descritos en el capítulo 5.

Adicionalmente, este capítulo presentó una comparación entre los mecanismos de restricciones débiles implementados en este trabajo y la reificación de restricciones, una forma muy común de resolver situaciones sobre-restringidas en Mozart. La diferencia conceptual entre ambos, que consiste en la noción de satisfacción parcial de restricciones que es inducida por la reificación, fue el principal objeto de comparación y al mismo tiempo, el principal impedimento para un análisis más profundo.

7 Consideraciones Finales

Este capítulo concluye el presente trabajo de grado en tres formas. Primero, las conclusiones principales del trabajo son recopiladas y sustentadas. Segundo, los resultados obtenidos y las contribuciones propuestas son comparadas con las propuestas más relevantes en la literatura. Finalmente, a modo de recomendaciones, se señalan algunas direcciones interesantes de trabajo futuro en el área de modelamiento y solución de problemas de satisfacción de restricciones débiles.

7.1. Conclusiones

1. Las direcciones más sólidas para la inclusión de conceptos de *debilidad* en el contexto de los problemas de satisfacción de restricciones son provistas por los denominados modelos formales o algebraicos. En este tipo de modelos se asumen esquemas *explícitos* de almacenamiento de las valuaciones relacionadas con las restricciones (o en su defecto, con las tuplas de valores que las representan). En otras palabras, la formulación teórica de los modelos ignora el problema del manejo eficiente de las valuaciones, suponiendo que su almacenamiento, consulta y modificación es, por lo menos, económico en términos computacionales. Esta perspectiva optimista del manejo de las valuaciones es evidente en la mayoría de los trabajos teóricos relacionados con restricciones débiles, y es, a nuestro juicio, uno de los aspectos más sobresalientes en la brecha existente entre teoría y práctica en el campo de las restricciones débiles.

2. El hecho de que en el esquema de abstracción estudiado la topología del problema (variables y restricciones) permanece intacta fue fundamental, porque restringió una parte significativa del trabajo al diseño de mecanismos para la transformación de las valuaciones asociadas con las restricciones. Tal diseño, influido también por el problema del almacenamiento de valuaciones discutido arriba, tuvo como objetivo proponer medios "indirectos" de transformación de valuaciones que no están explícitamente almacenadas.

El mecanismo resultante para lograr dicha transformación fue el esquema de contrapartes fuertes propuesto en el capítulo 3.

3. Por su definición, las propuestas conceptuales para el módulo de restricciones débiles y el esquema iterativo de abstracción (presentadas en el capítulo 3) son lo suficientemente genéricas para ser implementadas en cualquier lenguaje de programación por restricciones que incluya explícitamente técnicas de consistencia local.

4. Los resultados teóricos permitieron delimitar claramente los alcances de las implementaciones proyectadas en este trabajo. Tanto las propiedades del modelo basado en semianillos como las del esquema de abstracción fueron lo suficientemente específicas para este propósito. En este sentido, es posible identificar dos casos puntuales: por un lado, la idempotencia del operador multiplicativo de los semianillos difusos confirió seguridad para la implementación del módulo presentado en el capítulo 4, mientras que la preservación del orden en la abstracción de un CSP difuso con uno clásico permitió aprovechar varias propiedades fundamentales en una implementación. Adicionalmente, y exceptuando lo referente a la manipulación de las valuaciones, es posible afirmar que los modelos formales fueron aplicables en la práctica de forma sencilla.

5. Con respecto a la extensibilidad de los dos componentes implementados, si bien ambos están estructurados de forma tal que su extensión y/o mejora es posible, el diseño de nuevos propagadores que implementen restricciones débiles puede considerarse más sencillo que la definición de una contraparte fuerte para una restricción existente. Esta afirmación se justifica si se tiene en cuenta que la definición de una contraparte fuerte no es una labor trivial, en cuanto debe considerar criterios de simplicidad, eficiencia y coherencia con la restricción que se desea relajar. Adicionalmente, tal definición está condicionada —en mayor o menor medida— a la existencia de una restricción fuerte apropiada.

6. La relajación de un problema es una alternativa adecuada para obtener soluciones aproximadas en escenarios sobre-restringidos. La calidad de esta(s) solución(es) aproximada(s) dependerá de las condiciones particulares del problema y del método y/o técnica de relajación elegido. Un ejemplo de esta situación lo constituye la solución del problema de la fiesta progresiva, discutido en el capítulo 4.

7. De los resultados experimentales del módulo de restricciones débiles, es posible concluir que a medida que se aumenta el nivel de corte se encuentran menos soluciones de mayor

calidad. En general, es posible afirmar que el uso de restricciones débiles induce un aumento directo en el tamaño del problema. Consecuentemente, un aspecto fundamental en el uso de restricciones débiles es la relajación cuidadosa del problema, de forma que dicho aumento pueda ser controlado adecuadamente.

8. En concordancia con lo anterior, los medios para el control de los efectos de la la relajación resultan fundamentales para garantizar la viabilidad de los procesos de solución. Esto se ajusta bien a dos tipos de escenarios: el primero se presenta cuando se tiene un problema sobre-restringido y se desea obtener *alguna* solución. El segundo se da cuando teniendo soluciones (posiblemente aproximadas) se desea obtener soluciones adicionales. En ambos casos puede decirse que una estrategia controlada de relajación es altamente conveniente.

9. Tanto los resultados experimentales descritos en los capítulos 4 y 5 como la categorización de problemas presentada en el capítulo 6 sugieren que el esquema iterativo de abstracción se ajusta mejor a aquellos problemas en donde la mayoría de restricciones son débiles (problemas débiles fortalecidos), mientras en problemas fuertes relajados el uso de restricciones débiles resulta más conveniente. Relacionado con esta correspondencia, se encuentra el hecho de que la abstracción procura, en mayor o menor medida, encontrar soluciones óptimas, mientras que el módulo de restricciones débiles se concentra en encontrar soluciones, no necesariamente óptimas.

10. El modelo de relajación asociado con la idea de restricciones débiles considera todas las restricciones de un problema original, modificando los efectos que éstas tienen sobre los dominios de las variables relacionadas. Una ventaja de este tipo de relajación sobre los métodos basados en la satisfacción parcial de restricciones consiste en que, bajo las ideas de nuestra implementación, incluir una restricción es más conveniente (en términos de eficiencia) que removerla, pues de esa forma se garantiza algún grado de poda sobre los dominios de las variables. Esto se suma a otras ventajas de usar restricciones débiles (en lugar de satisfacción parcial), como las nociones concretas que se tienen del debilitamiento completo del problema y la generalidad de los criterios de selección de variables.

11. Una contribución significativa de este trabajo consiste en que propuso e implementó componentes de software que *amplían* las capacidades de un lenguaje de programación como Mozart. Esto representa una dirección contraria a la filosofía de otras implementaciones de mecanismos de restricciones, en donde el desarrollo de herramientas ad-hoc es una

práctica común.

12. El hecho de que Mozart tenga medios bien definidos para lograr implementaciones de bajo nivel fue una ventaja importante durante todo el proceso de implementación. Esto es especialmente significativo si se considera que la inclusión de dichos procedimientos en programas Mozart es relativamente sencillo. En particular, la definición y desarrollo del módulo de propagadores débiles y la implementación de la función de concretización se vieron ampliamente beneficiados de esta característica de Mozart. Sin embargo, algunos detalles de la implementación actual de Mozart impidieron un diseño más fiel del modelo original basado en semianillos. La pobre documentación del código fuente del lenguaje fue un obstáculo en este sentido, pues dificultó el desarrollo rápido de posibles alternativas de solución.

13. Los casos de estudio nos permiten concluir que la implementación de algunas de las restricciones débiles es susceptible a mejoras. Un punto fundamental aquí es que la implementación está diseñada de forma que mejoras y/o optimizaciones en el procesamiento de las restricciones son fácilmente integrables. De esta forma, el diseño de restricciones débiles eficientes se plantea como un reto a corto plazo.

14. Con respecto a los criterios de selección de soluciones aceptables, los modelos de restricciones débiles son más generales que los modelos de satisfacción parcial. Esto es evidente en técnicas como la reificación de restricciones, donde dichos criterios varían de problema a problema.

15. El hecho de que los componentes implementados son parametrizables abre un amplio abanico de posibilidades para su aplicación. Por ejemplo, algunos problemas pueden requerir criterios de distribución o técnicas de consistencia específicas; nuestra implementación es capaz de incluir tal tipo de mejoras. Adicionalmente, el esquema de abstracción también puede extenderse de manera similar.

7.2. Trabajo Relacionado

A continuación comparamos los resultados obtenidos en este trabajo con las propuestas teóricas y prácticas más relevantes.

Propuestas Teóricas

- *Teoría de conjuntos difusos ([KY95]).* Una noción que resulta similar a nuestra idea de nivel de corte es ampliamente utilizada en la teoría de conjuntos difusos. En dicho contexto, un α-*corte* de una función de pertenencia A (denotado por αA) es el conjunto de todos los x tales que $A(x)$ es mayor o igual que α. La relación entre esta idea y la noción de nivel de corte asociado con un CSP es inmediata. En nuestro caso, estamos interesados en en el conjunto (difuso) $\mathcal{U}$ de soluciones útiles. Así, τ representa el α-corte de la función de pertenencia del conjunto $\mathcal{U}$.

- *Restricciones de Alto Orden basadas en Semianillos ([Bis04], Capítulo 5)*: En este trabajo se presenta una definición funcional de los elementos del modelo basado en semianillos. Con tales definiciones es posible usar la composición, abstracción y aplicación de las funciones para establecer algoritmos de solución. De hecho, se propone un lenguaje funcional basado en expresiones y comandos para programar restricciones débiles. A manera de ejemplo, algoritmos de consistencia local son definidos en este lenguaje. A pesar de que el propósito de esta definición funcional es acercar los conceptos del modelo a un lenguaje implementable, la idea del almacenamiento explícito para las valuaciones permanece intacta. De esta forma, la desventaja analizada en el capítulo 3 y en la sección anterior persiste también.

- *Programación Concurrente por Restricciones Débiles ([Bis04], Capítulo 8).* En este trabajo se propone una forma de incluir las nociones del modelo basado en semianillos dentro del modelo original de programación concurrente por restricciones (cc) [SRP91]. La única adición consiste en el uso de una función que transforma niveles de preferencia en una noción absoluta (si/no) de consistencia. Adicionalmente, se considera una extensión de la sintaxis y la semántica operacional de cc de modo que los niveles del semianillo puedan manejarse. La principal diferencia consiste en que los agentes cuentan con un umbral de preferencia que es usado para determinar su estado (éxito, fallo, suspensión) y para podar los procesos de búsqueda.

- Al igual que Petit et al. [PRB01] creemos que en el diseño de algoritmos de consistencia para situaciones sobre-restringidas es fundamental considerar la semántica de cada restricción, particularmente en términos de eficiencia. De hecho, [PRB01] propone algoritmos específicos para el caso particular de la restricción *all-different*. Las nociones básicas de los algoritmos, basadas en términos de grafos, pueden integrarse de forma inmediata con las ideas propuestas en el capítulo 3 como funciones de consistencia y de

valuación.

Propuestas Prácticas

- [KO04] es quizás el trabajo que guarda más semejanzas con la idea de almacenamiento implícito de valuaciones desarrollada en el capítulo 3. Con base en la definición funcional del modelo de semianillos (discutida arriba), tal trabajo propone un esquema de *evaluación* de meta-restricciones, o funciones sobre restricciones. Las operaciones de combinación y proyección del modelo basado en semianillos (Definición 2.4) son definidas como meta-restricciones, por lo que la solución de un SCSP depende de estas meta-restricciones y no de algoritmos de consistencia local.

  La ventaja del acercamiento propuesto en [KO04] es que es una opción eficiente para la solución de problemas dados en términos de semianillos con operadores multiplicativos no idempotentes. La clave de esta eficiencia está en obtener las valuaciones de las tuplas solamente cuando es necesario, tal y como es discutido en el capítulo 3 e implementado en el módulo descrito en el capítulo 4. La principal diferencia de [KO04] con respecto a nuestro trabajo es que se trata de una técnica aislada, sin soporte de un lenguaje de programación. El módulo descrito en el capítulo 4 está directamente ligado con las estructuras y sintaxis de Mozart, con posibilidades claras de extensión y mantenimiento de las restricciones implementadas, ofreciendo ahorros (en espacio y eficiencia de los algoritmos) comparables a los mencionados en [KO04].

- *clp(FD,S) [GC98].* `clp(FD,S)` es un lenguaje que soporta el modelo de restricciones débiles basadas en semianillos en el contexto de la programación lógica por restricciones. La implementación de dicho lenguaje —sin soporte desde principios de 2002— aprovecha la extensión del concepto de arco-consistencia descrita en [Bis04]. El kernel del lenguaje, denominado sistema SFD, es completamente genérico con respecto al semianillo. El trabajo en [GC98] y la implementación descrita en el capítulo 4 comparten la idea de incrementalidad de un lenguaje de programación *existente* con nuevos elementos de restricciones débiles. Las restricciones que [GC98] provee (aritméticas y relacionales, esencialmente) resultan similares a las descritas en el capítulo 4. Es posible incluir nuevos semianillos de forma sencilla, manteniendo intacto el núcleo del sistema. Las pruebas de rendimiento presentadas en [GC98] se ocupan de resolver problemas con pocas variables, comparados con los problemas resueltos en los capítulos 4 y 5.

- *Resultados prácticos en la abstracción de restricciones débiles [BRP03].* Se trata de la

referencia más similar al esquema de abstracción presentado en este trabajo. En términos generales, se trata de una propuesta aislada de implementación con características limitadas. Más específicamente, identificamos las siguientes diferencias entre las dos propuestas:

1. La implementación descrita en el capítulo 5 y el sistema propuesto en [BRP03] son completamente diferentes. Nuestro esquema únicamente almacena un valor por cada restricción (el factor de penalización), una alternativa económica comparada con los costos de implementar las ideas en [BRP03], en donde se almacenan y manipulan las valuaciones asociadas con cada tupla en un SCSP. A diferencia de [BRP03], nuestra propuesta constituye una técnica de programación en si misma, que *extiende* un lenguaje de programación existente. Un aspecto relacionado que resulta particularmente significativo es que nuestra implementación del esquema de abstracción y el solucionador de problemas difusos descrito en el capítulo 4 comparten la misma sintaxis, lo cual difiere de la estrategia en [BRP03], en donde se utiliza un solucionador externo para problemas difusos (CON'FLEX [VGMC$^+$98]). Finalmente, los procedimientos de abstracción implementados son flexibles por la existencia de medios estándar para modificar y extender el conjunto de contrapartes fuertes. A nuestro mejor saber, estas capacidades no están disponibles en el sistema descrito en [BRP03].
2. Con respecto al procedimiento iterativo, las opciones disponibles en el algoritmo 5.1 son similares a las tres versiones del algoritmo propuesto en [BRP03], denominadas A1, A2 y A3. Sin embargo, los modos del algoritmo 5.1 ofrecen características adicionales que pueden mejorar los procesos de solución. Primero, en nuestros procedimientos de abstracción, el usuario puede especificar la precisión deseada del intervalo de soluciones, así como un valor para el mínimo nivel de tolerancia (i.e., t). Ninguna de estas opciones está disponible para el esquema descrito en [BRP03], donde únicamente una precisión fija de 1/10 existe en el algoritmo A1. Como el algoritmo A2 en [BRP03], el modo ávido considera la valuación de la mejor solución encontrada para calcular el nuevo límite inferior del algoritmo. No obstante, nuestra versión del modo ávido también tiene en cuenta el límite dado por el modo binario, pues éste último puede ser mejor que el límite dado por la valuación de la mejor solución encontrada hasta el momento. Finalmente, el modo pesimista constituye una mejora del algoritmo A3, si se considera que cuando no encuentra solución, la búsqueda de una nueva solución *continúa*, usando la tolerancia mínima

dada por el usuario como límite inferior. Por su parte, A3 se detiene tan pronto no encuentre una solución.

3. Si bien el rendimiento en tiempo es similar en ambos sistemas, dos aspectos deben considerarse. Primero, como se mencionaba antes, el nivel de precisión del algoritmo utilizado en [BRP03] es pequeño comparado con el nivel utilizado en nuestros experimentos. Claramente, esta precisión tiene influencia directa sobre el número de iteraciones requeridas para un problema, lo que a su vez, tiene influencia en el rendimiento completo.

7.3. Recomendaciones

Creemos que existen varias direcciones en las cuales las ideas principales de esta tesis pueden ser ampliadas. Algunas de ellas se señalan a continuación.

1. Implementar los conceptos propuestos en el capítulo 3 en otro solucionador de restricciones para probar su generalidad y ortogonalidad. Por ejemplo, una implementación en GECODE [TS05] puede ser interesante, pues su estructura de librería extensible podría ayudar a que un mayor número de aplicaciones prácticas hagan uso de las contribuciones propuestas.

2. Realizar un estudio sistemático de la aplicabilidad, alcances y falencias de las implementaciones propuestas, con el objetivo de mejorar su rendimiento en aplicaciones prácticas.

3. Ampliar la cantidad de propagadores débiles disponibles para Mozart, de tal forma que se aumente el rango de problemas que pueden ser modelados utilizando restricciones débiles y demostrando en diversos campos la utilidad de este enfoque a la hora de modelar con mayor fidelidad los problemas de la vida real.

4. Realizar estudios e implementaciones de métodos adaptativos que permitan realizar procesos de distribución más eficientes y eficaces. Estas mejoras se verían reflejadas en el aumento de la eficacia de los propagadores a la hora de podar los dominios de las variables.

5. Directamente relacionado con el punto anterior se encuentra la idea de continuar la exploración y el desarrollo de criterios de selección de las variables a distribuir, así como

el estudio de criterios que permitan, una vez seleccionada la variable, seleccionar el mejor valor de su dominio de modo que el proceso de propagación se vea beneficiado.

6. El semianillo con el que se trabaja en esta tesis tiene un operador de combinación idempotente, lo que facilita la implementación de técnicas de consistencia local que son determinantes en el proceso de búsqueda de soluciones. Los semianillos con operadores de combinación no idempotentes son utilizados para modelar un amplio campo de problemas, pero su naturaleza hace que las técnicas de consistencia local no sean aplicables, aumentando notablemente la complejidad de la búsqueda de solución. Cooper y Schiex [CS04] han propuesto un esquema de restricciones débiles que puede modelar satisfactoriamente técnicas de consistencia local para operadores no idempotentes. Incluso, en un trabajo reciente, dicha propuesta ha sido asociada a la idea de límite consistencia [GSZ05]. Estos estudios podrían considerarse como base para realizar una implementación eficiente de restricciones débiles basadas en semianillos con operadores no idempotentes, ampliando así significativamente el rango de problemas que podrían modelarse con restricciones débiles.

7. Los estudios descritos en el punto anterior también pueden ser la base de un esquema de abstracción entre semianillos con operadores idempotentes y no idempotentes, lo que permitiría, por ejemplo, la solución de un semianillo de optimización utilizando un semianillo difuso o uno clásico.

8. Estudiar las propuestas de Petit et al. [PRB01] con respecto a implementación de algoritmos de consistencia para resolver problemas sobre-restringidos, con el objetivo de mejorar el rendimiento de los propagadores estudiando la semántica propia de cada restricción.

Bibliografía

[Apt03] Krzysztof R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.

[Bar99] R. Barták. Constraint Programming: What is behind? In *Proceedings of CPDC99 Workshop (Invited Talk)*, June 1999.

[BCR02] Stefano Bistarelli, Philippe Codognet, and Francesca Rossi. Abstracting soft constraints: framework, properties, examples. *Artif. Intell.*, 139(2):175–211, 2002.

[BFM+96] Stefano Bistarelli, Helene Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gerard Verfaillie. Semiring-based csps and valued csps: Basic properties and comparison. In *Over-Constrained Systems*, pages 111–150, London, UK, 1996. Springer-Verlag.

[Bis04] Stefano Bistarelli. *Semirings for Soft Constraint Solving and Programming.* Number 2962 in Lecture Notes in Computer Science. Springer-Verlag, 2004.

[BMH94] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS 94 - MIT Press*, pages 1–21, 1994.

[BRP03] Stefano Bistarelli, Francesca Rossi, and Isabella Pilan. Abstracting soft constraints: Some experimental results on fuzzy csps. In *Recent Advances in Constraints*, volume 3010 of *LNCS*, pages 107–123. Springer, 2003.

[CdGL+99] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio link frequency assignment. *Constraints: An International Journal*, 4(1), 1999.

[CFMM05] Carlos Castro, Christian Figueroa, Eric Monfroy, and Rafael Meneses. Constraint Solving Using Dynamic Variable and Value Selection. In Juan F. Díaz, Camilo Rueda, and Antal A. Buss, editors, *Proceedings of CLEI2005*, pages 19–28, October 2005. ISBN 958-670-426-2.

[Cou96] P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, June 1996.

[CS04] Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artif. Intell.*, 154(1-2):199–227, 2004.

[Dec03] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[DFP93] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.

[DOPR04] Alberto Delgado, Carlos A. Olarte, Jorge Andrés Pérez, and Camilo Rueda. Implementing Semiring-Based Constraints using a Concurrent Constraint Programming Language. In Stefano Bistarelli and Francesca Rossi, editors, *Proceedings of the Sixth International Workshop on Preferences and Soft Constraints (Soft 2004)*, October 2004. Held in conjunction with the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004).

[DOPR05a] Alberto Delgado, Carlos A. Olarte, Jorge Andrés Pérez, and Camilo Rueda. Implementing Semiring-Based Constraints Using Mozart. In Peter Van Roy, editor, *Multiparadigm Programming in Mozart/Oz: Extended Proceedings of the Second International Conference MOZ 2004*, volume 3389 of *Lecture Notes in Computer Science*, pages 224–236. Springer-Verlag, 2005.

[DOPR05b] Alberto Delgado, Carlos A. Olarte, Jorge Andrés Pérez, and Camilo Rueda. Semiring-based Fuzzy Constraints in Concurrent Constraint Programming. In Juan F. Díaz, Camilo Rueda, and Antal A. Buss, editors, *Proceedings of CLEI2005*, pages 735–746, October 2005. ISBN 958-670-426-2.

[DPP+05] Alberto Delgado, Jorge Andrés Pérez, Gustavo Pabón, Rafael Jordan, Juan F. Díaz, and Camilo Rueda. An Interactive Tool for the Controlled Execution of an Automated Timetabling Constraint Engine. In *Multiparadigm Programming in Mozart/Oz*, volume 3389 of *LNCS*. Springer-Verlag, 2005.

[DPR05] Alberto Delgado, Jorge Andrés Pérez, and Camilo Rueda. Implementing an Abstraction Framework for Soft Constraints. In Jean-Daniel Zucker and Lorenza Zaitta, editors, *Abstraction, Reformulation, and Approximation, Proceedings of the 6th International Symposium SARA 2005*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 60–75. Springer, July 2005.

[FA03] Thom Frühwirth and Slim Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.

[FL93] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, number 747 in LNCS, pages 97–104. Springer-Verlag, 1993.

[Fre89] Eugene C. Freuder. Partial Constraint Satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA*, pages 278–283, 1989.

[GC98] Yan Georget and Philippe Codognet. Compiling semiring-based constraints with clp(fd,s). In *Proceedings of CP'98*, 1998.

[GSZ05] Christine Gaspin, Thomas Schiex, and Matthias Zytnicki. Bound arc consistency for weighted CSPs. In Simon de Givry and Weixiong Zhang, editors, *Proceedings of the Seventh International Workshop on Preferences and Soft Constraints (Soft 2005)*, October 2005. Held in conjunction with the Eleventh International Conference on Principles and Practice of Constraint Programming (CP 2005).

[KO04] Jerome Kelleher and Barry O'Sullivan. Evaluation-based semiring meta-constraints. In *Proceedings of MICAI*, volume 2972 of *Lecture Notes in Computer Science*. Springer, April 2004.

[KY95] George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[MÖ1] Tobias Müller. *Constraint Propagation in Mozart*. PhD thesis, Universitat des Saarlandes, 2001.

[Mac77] A. Mackworth. Consistency in networks of relations. *Journal of Artificial Intelligence*, 8(1), 1977.

[MF02] Zbigniew Michalewicz and David B Fogel. *How to Solve It: Modern Heuristics*. Springer, 2002.

[Mon74] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

[MS98] K. Marriot and Peter J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[PRB01] T. Petit, J.C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *Proc. of CP'2001*, volume 2239 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[RPD90] Francesca Rossi, Charles J. Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *ECAI*, pages 550–556, 1990.

[RS98] Francesca Rossi and Alessandro Sperduti. Learning solution preferences in constraint problems. *J. Exp. Theor. Artif. Intell.*, 10(1):103–116, 1998.

[SBHW95] B. M. Smith, S. C. Brailsford, P. M. Hubbard, and H. P. Williams. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. Technical Report 95.8, University of Leeds, School of Computer Studies, March 1995.

[SFV95] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. IJCAI95*, pages 631–637, San Francisco, CA - USA, 1995. Morgan Kauffman.

[Smo95] Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000, pages 324–343. Berlin, 1995.

[SRP91] V. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *18th Annual ACM Symposium on Principles of Programming Languages*, pages 333–353. ACM Press, 1991.

[SS04] Christian Schulte and Gert Smolka. *Finite Domain Constraint Programming in Oz. A Tutorial*, 2004. Available at `www.mozart-oz.org`.

[TS05] Guido Tack and Christian Schulte. GECODE: Generic Constraint Development Environment, 2005. More information available at `www.gecode.org`.

[VGMC+98] Frédéric Vardon, Christine Gaspin, Roger Martin-Clouaire, Thomas Schiex, Martin Bouchez, Patrick Chabrier, and Marie-Françoise Jourjon. Con'flex, un logiciel de resolution de problemes de satisfaction de contraintes. INRA (Institut National de la Recherche Agronomique) web site, 1998. See `http://www.inra.fr/bia/T/conflex` for more information.