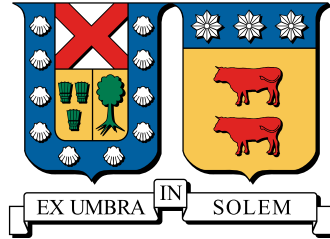


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO - CHILE



**DISTINTOS ENFOQUES DE MODELAMIENTO
MATEMÁTICO PARA RESOLVER PROBLEMAS
DE OPTIMIZACIÓN USANDO *OPTIMIZATION
MODULO THEORIES***

IGNACIO ALEJANDRO JORQUERA NAVARRO

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

PROFESOR GUÍA: NICOLÁS SEBASTIÁN GÁLVEZ RAMÍREZ
PROFESOR CORREFERENTE: CARLOS CASTRO

OCTUBRE - 2023

Quiero dedicar este trabajo a mi familia, en especial a mi madre por el infinito apoyo
que me ha dado durante los 27 años que me tomó llegar hasta aquí.

Sin ti, esto no habría sido posible.

Gracias.

AGRADECIMIENTOS

¡Último día, nadie se enoja! Fueron varios años llenos de altos y bajos, de situaciones inesperadas y de mucho que hacer, pero con esto cierra una de las etapas más importantes de mi vida (hasta ahora, al menos).

Quiero agradecer a mi increíble equipo de investigación, a mis tutores, Nicolás Gálvez Ramírez y Carlos Castro, y a mis pares, Raimundo Gross y Nicolás Cerpa. Me dieron el apoyo necesario para dar cierre a esta etapa y la oportunidad para seguir avanzando más allá en mi carrera. Espero poder seguir contando con ustedes en lo que dure nuestro recorrido por el postgrado.

De paso, también quiero agradecer a todas las personas con las que compartí sudor y lágrimas durante los casi 6 años que duró este periodo, Juan Pablo y Patricio siempre dispuestos a salvar el día desde el primer semestre de la carrera, Héctor, Anastasiia, Tamara, Nicolás, sobrevivir a feria de software y a los últimos ramos hubiese sido terrible sin todo el apoyo de ustedes, y a todas las personas con las que he tenido la oportunidad llevar a cabo proyectos y tareas.

RESUMEN

La optimización ha sido una parte fundamental de la humanidad desde sus orígenes, ya sea para administrar recursos o resolver problemas cotidianos. En esta tesis se abordan algunos problemas de optimización lineal y el impacto que tiene la decisión de diseño de elegir una lógica o dominio al momento de construir un modelo matemático. En particular, se modelan los problemas *N-queens*, *Travelling Salesman Problem (TSP)*, *Nurse Scheduling Problem (NSP)*, *Balanced Academic Curriculum Problem (BACP)* y *Unbounded Knapsack Problem (UKP)* usando las lógicas *Closed linear formulas over Linear Integer Arithmetic (LIA)*, *Closed linear formulas in Linear Real Arithmetic (LRA)*, *Closed Quantifier-Free formulas over the theory of fixed-size BitVectors (QF_BV)* y *Closed Quantifier-Free formulas over the theory of Integer Arrays (QF_ALIA)* definidas en la *SMT-LIB*, y luego se resuelven usando el *solver* para *Optimization Modulo Theories (OMT)* de *Z3* con el objetivo de analizar y comparar cualitativa y cuantitativamente distintos enfoques matemáticos para conocer los beneficios y desventajas que presentan al momento de resolver problemas de *OMT* según el tipo de problema.

En particular se busca:

- Modelar y resolver usando *OMT* al menos 3 problemas de satisfacción de restricciones usando al menos 3 lógicas distintas de *Satisfiability Modulo Theories (SMT)*.
- Estudiar la opción de combinar distintas lógicas en un mismo problema para construir nuevos casos de estudio.
- Evaluar y comparar el rendimiento de los problemas resueltos.

Dentro de lo logrado con esta propuesta se presenta evidencia empírica sobre diferencias en el desempeño al resolver *Constraint Satisfaction Problems (CSP)* y *Constraint Satisfaction and Optimization Problems (CSOP)* usando modelos basados en las lógicas mencionadas, sin embargo, la falta de patrones claros para determinar que lógica usar dependiendo del problema hace que este se vuelva un problema de ajustar parámetros para ver que lógica entrega mayores beneficios para el problema que se quiera resolver. También se presentan reglas para traducir problemas modelados con lógica *LIA* a *LRA*, *QF_BV* y *QF_ALIA*, lo que permite probar rápidamente cambios de dominio en modelos ya existentes para determinados *CSP/CSOP*.

Palabras Clave—*Satisfiability Modulo Theories*, *Optimization Modulo Theories*, Problemas de satisfacción de restricciones, *Z3*

ABSTRACT

Optimization has been a fundamental part of humanity since its origins, whether to manage resources or solve everyday problems. This thesis addresses some linear optimization problems and the impact of the design decision to choose a logic or domain when building a mathematical model. In particular, the problems *N-queens*, *Travelling Salesman Problem (TSP)*, *Nurse Scheduling Problem (NSP)*, *Balanced Academic Curriculum Problem (BACP)* and *Unbounded Knapsack Problem (UKP)* using the logics *Closed linear formulas over Linear Integer Arithmetic (LIA)*, *Closed linear formulas in Linear Real Arithmetic (LRA)*, *Closed Quantifier-Free formulas over the theory of fixed-size BitVectors (QF_BV)* and *Closed Quantifier-Free formulas over the theory of Integer Arrays (QF_ALIA)* defined in the *SMT-LIB*, and then solved using the solver for *Optimization Modulo Theories (OMT)* of *Z3* to analyze and compare qualitatively and quantitatively different mathematical approaches to know the benefits and disadvantages they present when solving *OMT* problems according to the type of problem.

In particular, we are looking for:

- Model and solve using *OMT* at least 3 constraint satisfaction problems using at least 3 different logics from *Satisfiability Modulo Theories (SMT)*.
- Analyze the option of mixing different *SMT* logics in the same problem to build new case studies.
- Evaluate and compare the performance of the solved problems.

Within what was achieved with this proposal, empirical evidence is presented on differences in performance when solving *Constraint Satisfaction Problems (CSP)* and *Constraint Satisfaction and Optimization Problems (CSOP)* using models based on the logic mentioned above, however, the lack of clear patterns to determine which logic to use depending on the problem makes this become a problem of adjusting parameters to see which logic provides greater benefits for the problem to be solved. Rules for translating problems modeled with *LIA* logic to *LRA*, *QF_BV*, and *QF_ALIA* are also presented, allowing you to quickly test domain changes in existing models for certain *CSP/CSOP*.

Keywords—Satisfiability Modulo Theories, Optimization Modulo Theories, Constraint Satisfaction Problems, Z3

Glosario

AX: *ArraysEx*.

BACP: *Balanced Academic Curriculum Problem*.

BV: *FixedSizeBitVectors*.

CP: *Constraint Programming*.

CSP: *Constraint Satisfaction Problem*.

CSOP: *Constraint Satisfaction and Optimization Problem*.

LIA: *Linear Integer Arithmetic*.

LRA: *Linear Real Arithmetic*.

NSP: *Nurse Scheduling Problem*.

OMT: *Optimization Modulo Theories*.

SMT: *Satisfiability Modulo Theories*.

TSP: *Travelling Salesman Problem*.

UKP: *Unbounded Knapsack Problem*.

Z3: *SMT solver de Microsoft*.

Índice general

Índice de figuras	IV
Índice de tablas	VI
1. Introducción	1
1.1. Objetivos	2
1.1.1. Objetivo General	2
1.1.2. Objetivos Específicos	2
1.2. Contribuciones	2
1.3. Estructura del informe	3
2. Modelamiento, ¿un problema a considerar?	4
2.1. Programación lineal	4
2.1.1. Programación lineal entera	5
2.2. <i>SAT</i>	7
2.3. <i>SMT</i>	8
2.3.1. Aplicaciones	11
2.4. <i>Optimization Modulo Theories</i> (<i>OMT</i>)	12
2.5. Definición y formalización del problema	12
3. Propuesta de solución	14
3.1. Problemas a analizar	14

3.1.1. Instancias	16
3.2. Modelamiento inicial de problemas usando LIA	20
3.2.1. <i>N-queens</i>	20
3.2.2. <i>Balanced Academic Curriculum Problem (BACP)</i>	21
3.2.3. <i>Nurse Scheduling Problem (NSP)</i>	23
3.2.4. <i>Travelling Salesman Problem (TSP)</i>	26
3.2.5. <i>Unbounded Knapsack Problem (UKP)</i>	27
3.3. Tipos de restricciones	28
3.4. Traducción del modelamiento hacia otras lógicas <i>SMT</i>	29
3.4.1. Transformación de LIA a LRA	29
3.4.2. Transformación de LIA a QF_BV	32
3.4.3. Transformación de LIA a QF_ALIA	35
4. Resultados	38
4.1. Optimización de modelos y aspectos técnicos	38
4.2. Calidad de las soluciones encontradas	39
4.3. Tiempo empleado en encontrar solución	42
4.4. Uso de memoria	45
5. Conclusiones	48
5.1. Trabajos futuros	51
Anexo	53
Transformaciones de LIA a LRA	53
<i>N-queens</i>	53
<i>Balanced Academic Curriculum Problem (BACP)</i>	54
<i>Nurse Scheduling Problem (NSP)</i>	56
<i>Unbounded Knapsack Problem (UKP)</i>	58
Transformaciones de LIA a QF_BV	59
<i>N-queens</i>	59
<i>Balanced Academic Curriculum Problem (BACP)</i>	60

<i>Nurse Scheduling Problem (NSP)</i>	62
<i>Unbounded Knapsack Problem (UKP)</i>	65
Transformaciones de LIA a QF_ALIA	66
<i>N-queens</i>	66
<i>Balanced Academic Curriculum Problem (BACP)</i>	67
<i>Nurse Scheduling Problem (NSP)</i>	68
<i>Unbounded Knapsack Problem (UKP)</i>	71
Tablas con información adicional recopilada	71
References	74

Índice de figuras

2.1.	Figura con una de las posibles soluciones al problema <i>N-Queens</i>	5
2.2.	Diagrama con las lógicas <i>SMT</i>	10
4.1.	Gráfico con los tiempos totales de ejecución por instancia de <i>N-queens</i> .	42
4.2.	Gráfico con los tiempos totales de ejecución por instancia de <i>NSP</i> . . .	43
4.3.	Gráfico con los tiempos totales de ejecución por instancia de <i>UKP</i> . . .	43
4.4.	Gráfico con los tiempos totales de ejecución por instancia de <i>TSP</i> . . .	44
4.5.	Gráfico con los tiempos totales de ejecución por instancia de <i>BACP</i> . .	45

Índice de tablas

4.1.	Tabla con las soluciones obtenidas para <i>NSP</i>	40
4.2.	Tabla con las soluciones obtenidas para <i>N-queens</i>	40
4.3.	Tabla con las soluciones obtenidas para <i>UKP</i>	41
4.4.	Tabla con las soluciones obtenidas para <i>TSP</i>	41
4.5.	Tabla con las soluciones obtenidas para <i>BACP</i>	42
4.6.	Tabla con el consumo máximo de RAM para <i>BACP</i>	45
4.7.	Tabla con el consumo máximo de RAM para <i>TSP</i>	46
4.8.	Tabla con el consumo máximo de RAM para <i>NSP</i>	46
4.9.	Tabla con el consumo máximo de RAM para <i>UKP</i>	47
4.10.	Tabla con el consumo máximo de RAM para <i>N-Queens</i>	47
5.1.	Tabla con el tiempo total de ejecución para <i>N-queens</i>	72
5.2.	Tabla con el tiempo total de ejecución para <i>NSP</i>	72
5.3.	Tabla con el tiempo total de ejecución para <i>UKP</i>	72
5.4.	Tabla con el tiempo total de ejecución para <i>TSP</i>	73
5.5.	Tabla con el tiempo total de ejecución para <i>BACP</i>	73

Introducción

Optimization Modulo Theories (OMT) es un *framework* [1] cuya función es la de extender *Satisfiability Modulo Theories (SMT)* y con ello lograr resolver *Constraint Satisfaction and Optimization Problems (CSOP)* modelados a partir de distintas teorías, vale decir, lógicas de primer-orden y sus combinaciones *SMT*.

Algunas de las aplicaciones de *SMT* más importantes incluyen la construcción de *Interactive Theorem Prover*, los cuales buscan verificar la correctitud de una demostración matemática mediante un ciclo de interacción humano-máquina, siendo *HOL4* [2], uno de los *theorem prover* más destacados hoy en día.

Otro impacto relevante de *SMT* ocurre en los *Constraint Satisfaction Problems (CSP)*, área en la que trabajos como [3, 4] han demostrado algunos de los aportes que entrega *SMT* a problemas de *scheduling* y *planning*, respectivamente.

OMT aparece como una extensión que combina *SMT* con problemas de optimización, siendo una herramienta que permite optimizar problemas de satisfacción de restricciones con la flexibilidad que ofrece *SMT* para ampliar los dominios hacia múltiples lógicas producto de combinaciones entre las teorías base que dispone *SMT* para trabajar en múltiples dominios, como por ejemplo: dominio de booleanos, de números enteros, números binarios, números reales, entre otros.

Usar *OMT* o *SMT* resulta atractivo cuando hay problemas de satisfacción de restricciones que requieren un tipo de lógica en específico para ser resueltos, como es el caso particular de algunas tareas de verificación de *hardware/software* que se desarrollan en entornos donde solo se pueden realizar operaciones entre *bit-vectors* [5].

Pero también hay casos en los que un determinado modelamiento de un problema puede tener distinto desempeño dependiendo de cuál lógica *SMT* es usada al momento de construir el modelo, es decir, resolver un problema particular usando solo lógica de *bit-vectors* podría presentar un mejor desempeño que resolver el mismo problema con

lógica de enteros [6], y esto abre la posibilidad de que se repitan escenarios en donde, dependiendo del tipo de problema, modelos basados en determinadas lógicas lleven a cabo una mejor búsqueda de la solución óptima.

1.1. Objetivos

Esta propuesta busca comparar enfoques matemáticos usando distintas lógicas y teorías *SMT* al momento de modelar y resolver determinados problemas de satisfacción de restricciones y/o optimización utilizando *OMT*.

Los distintos enfoques o modelamientos matemáticos se utilizan para representar cada restricción de un problema en distintas lógicas *SMT*

Y para ello se plantean los siguientes objetivos generales y específicos:

1.1.1. Objetivo General

- Analizar y comparar cualitativa y cuantitativamente distintos enfoques matemáticos para conocer los beneficios y las desventajas de estos según el tipo de problema analizado y el desempeño del modelo al resolverlo usando *OMT*.

1.1.2. Objetivos Específicos

- Modelar y resolver usando *OMT* al menos 3 problemas de satisfacción de restricciones usando al menos 3 lógicas distintas de *Satisfiability Modulo Theories*.
- Estudiar la opción de combinar distintas lógicas en un mismo problema para construir nuevos casos de estudio.
- Evaluar y comparar el rendimiento de los problemas resueltos.

1.2. Contribuciones

En cuanto a las contribuciones de esta tesis, se presenta evidencia empírica de que modelar problemas *CSP/CSOP* usando distintas lógicas *SMT* impacta directamente en el desempeño de los modelos, específicamente en la calidad de las soluciones encontradas, el tiempo de ejecución que demora la búsqueda de soluciones y en la cantidad de recursos consumidos a lo largo de la ejecución. También se presenta una serie de reglas para traducir modelos de programación lineal entera a 3 lógicas *SMT*: *LRA*, *QF_BV* y *QF_ALIA*, lo que permite cambiar el dominio de modelos ya existentes sin pérdida de generalidad.

1.3. Estructura del informe

La estructura de este documento está diseñada de la siguiente manera:

- En el Capítulo 1 se explica en detalle el contexto bajo el que se desarrolla esta tesis, incluyendo lo que se busca lograr y las contribuciones hechas.
- En el Capítulo 2 se revisa el estado del arte y a partir de este se define el problema a abordar en esta tesis.
- En el Capítulo 3 se abarca todo el procedimiento que hay por detrás de la propuesta de solución, desde los problemas seleccionados y sus instancias hasta el modelamiento de éstos usando distintas lógicas *SMT*.
- En el Capítulo 4 se explican las configuraciones usadas en el *solver OMT* y se lleva a cabo todo el proceso de análisis y validación de las soluciones encontradas.
- Finalmente, en el Capítulo 5 se presentan las conclusiones de esta tesis y se proponen trabajos futuros relevantes.
- Adicionalmente, en el Anexo se encuentran las traducciones de los modelos que no se incluyen en el documento principal.

CAPÍTULO 2

Modelamiento, ¿un problema a considerar?

En este capítulo se estudian diversas formas de modelamiento y cómo, a modo de estado del arte, han afectado a la resolución de problemas de satisfacción de restricciones y optimización, para luego formalizar la problemática abordada en la presente memoria.

2.1. Programación lineal

La programación lineal (también conocida como optimización lineal) es un método o técnica matemática que permite maximizar o minimizar una función lineal con una o más variables. A base de lo explicado en [7], ésta se caracteriza por modelar situaciones de la realidad mediante combinaciones lineales de las variables de decisión.

Al modelar una situación o problema usando programación lineal se deben considerar los siguientes elementos claves:

- **Función objetivo:** Medida o métrica cuantitativa del funcionamiento del sistema que se desea optimizar.
- **Parámetros:** Valores constantes conocidos, son definidos por el contexto del problema.
- **Variables:** Decisiones que se pueden tomar para afectar el valor de la función objetivo.
- **Dominios:** Posibles valores de las variables y sub-índices.
- **Restricciones:** Relaciones (ecuaciones e inecuaciones) que las variables están obligadas a cumplir.

El dominio de las variables problema da lugar a casos específicos de programación lineal. Generalmente se definen en los números reales, pero si se acota el dominio únicamente a los números enteros se produce la variante conocida como *Programación Lineal Entera*, como se ve a continuación.

2.1.1. Programación lineal entera

Una variación de la programación lineal es la programación lineal entera, la cual requiere que las variables del problema tengan su dominio en los números enteros.

- Si todas las variables del problema son de tipo entero, entonces este pertenece a problemas de *Programación Lineal Entera* pura.
- Si solo algunas de las variables del problema son de tipo entero, entonces este pertenece a problemas de *Programación Lineal Entera Mixta*.
- Si las variables del problema son binarias (dominio $\{0, 1\}$), entonces éste pertenece a problemas de *Programación Lineal Binaria*.

A modo de ejemplo, consideremos el problema 054 de la *CSPLib*: *N-Queens* [8]:

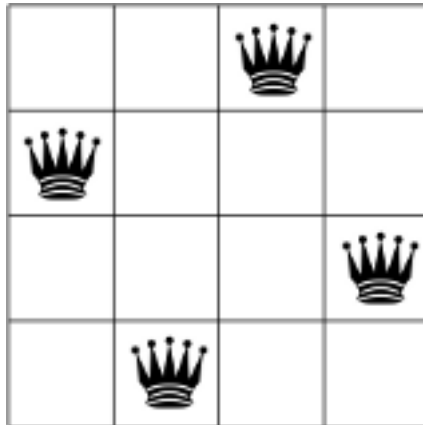


Figura 2.1: Tablero mostrando una de las posibles soluciones al problema *N-Queens* cuando $N = 4$. Fuente: <https://developers.google.com/optimization/cp/queens>.

La pregunta que busca resolver este problema es: ¿Pueden N reinas del mismo color ser posicionadas en un tablero de ajedrez de tamaño $N \times N$, tal que ninguna de las reinas pueda atacar o ser atacada por las otras? Notar que una reina será atacada si hay otra reina en la misma fila, columna o diagonal.

Este problema puede ser modelado siguiendo el ejemplo presentado en [9] usando programación lineal entera:

- **Función objetivo:**

Minimizar el número de reinas atacadas.

- **Parámetros:**

N : Número de reinas y número de filas y columnas del tablero, $N \in \mathbb{Z}^+$

- **Variables:**

$$X_{ij} = \begin{cases} 1 & \text{Si hay una reina en la posición } (i, j), \\ 0 & \text{en caso contrario, } \forall i, j \in \{1, \dots, N\} \end{cases}$$

i : índice de las filas

j : índice de las columnas

- **Restricciones:**

1. Debe haber N reinas en el tablero:

$$\sum_{i=1}^N \sum_{j=1}^N X_{ij} = N$$

2. Debe haber a lo más una reina por columna:

$$\sum_{j=1}^N X_{ij} \leq 1, \forall i \in \{1, \dots, N\}$$

3. Debe haber a lo más una reina por fila:

$$\sum_{i=1}^N X_{ij} \leq 1, \forall j \in \{1, \dots, N\}$$

4. Debe haber a lo más una reina por diagonal positiva:

$$\sum_{i=1}^N \sum_{j=1}^N X_{ij} \leq 1 : i - j = k, \forall k \in \{2 - n, \dots, n - 2\}$$

5. Debe haber a lo más una reina por diagonal negativa:

$$\sum_{i=1}^N \sum_{j=1}^N X_{ij} \leq 1 : i + j = k, \forall k \in \{3, \dots, 2n - 1\}$$

Este tipo de *CSP* puede tener o no una función objetivo (se convierte en *CSOP* en caso de tenerla). En el ejemplo anterior se tiene que la función objetivo está implícita dentro de las restricciones, específicamente, en la cantidad de reinas por fila, columna y diagonales, razón por la que se omitió la combinación lineal explícita para representar la cantidad de reinas atacadas.

2.2. SAT

Los problemas de satisfacción booleana (o *SAT*, del inglés *Satisfiability*) buscan determinar si es que existe una asignación de valores para un conjunto de variable booleanas, tal que un conjunto de fórmulas proposicionales lógicas asociada al problema pueda ser reducida a una tautología (proposición verdadera o *True* en inglés) y se considera el motor base para abordar problemas de satisfacción de restricciones. Se ha demostrado que estos problemas tienen un nivel de complejidad NP-completo [10].

Este tipo de problemas se encuentra presente en distintas áreas de estudios, como por ejemplo en *model checking*, *AI planning*, testing de circuitos, modelamiento de *software*, y cualquier otra que pueda ser formulada como una instancia de *SAT* [11, 12, 13].

La notación para este tipo de problemas es la siguiente:

1. Átomo: fórmula booleana.
2. Si una variable $\mathcal{F}$ es una fórmula booleana, entonces $\neg\mathcal{F}$ es una fórmula booleana y es el complemento de $\mathcal{F}$.
 - Cuando $\mathcal{F}$ representa a un átomo x , entonces $\neg\mathcal{F}$ representa a $\bar{x}$.
 - Las variables de una fórmula booleana $\mathcal{F}$ son representadas como $var(\mathcal{F})$.
3. Si $\mathcal{F}$ y $\mathcal{G}$ son fórmulas booleanas, entonces $(\mathcal{F} \vee \mathcal{G})$ es una fórmula booleana.
4. Si $\mathcal{F}$ y $\mathcal{G}$ son fórmulas booleanas, entonces $(\mathcal{F} \wedge \mathcal{G})$ es una fórmula booleana.

De lo anterior también se puede extraer otras definiciones a partir de conectores lógicos, como por ejemplo, $\Rightarrow$ y/o $\Leftrightarrow$ para representar implicación lógica y relación bi-condicional de equivalencia, respectivamente.

Usando *SAT* es posible modelar el problema de las N reinas visto en la Sección 2.1.1 utilizando proposiciones lógicas [14]:

- **Parámetros:**

N : Número de reinas y número de filas y columnas del tablero, $N \in \mathbb{Z}^+$

- **Variables:**

$$X_{ij} = \begin{cases} True & \text{si hay una reina en la casilla } (i, j), \\ False & \text{en caso contrario, } \forall i, j \in \{1, \dots, N\} \end{cases}$$

i : índice de las filas

j : índice de las columnas

- **Restricciones:**

1. Debe haber N reinas en el tablero:

$$x_{i1} \vee x_{i2} \vee \dots \vee x_{iN} = \text{True}, \forall i \in \{1, \dots, N\}$$

2. No puede haber más de una reina por columna:

$$\neg ((x_{1j} \wedge x_{2j}) \vee (x_{1j} \wedge x_{3j}) \vee \dots \vee (x_{1j} \wedge x_{Nj}) \vee \\ (x_{2j} \wedge x_{3j}) \vee \dots \vee (x_{2j} \wedge x_{Nj}) \vee \\ \dots \vee (x_{(N-1)j} \wedge x_{Nj})) = \text{True}, \forall j \in \{1, \dots, N\}$$

3. No puede haber más de una reina por fila:

$$\neg ((x_{i1} \wedge x_{i2}) \vee (x_{i1} \wedge x_{i3}) \vee \dots \vee (x_{i1} \wedge x_{iN}) \vee \\ (x_{i2} \wedge x_{i3}) \vee \dots \vee (x_{i2} \wedge x_{iN}) \vee \\ \dots \vee (x_{i(N-1)} \wedge x_{iN})) = \text{True}, \forall i \in \{1, \dots, N\}$$

4. No puede haber más de una reina por diagonal:

$$\neg ((x_{00} \wedge x_{11}) \vee (x_{00} \wedge x_{22}) \dots (x_{00} \wedge x_{NN}) \vee \\ (x_{11} \wedge x_{22}) \vee \dots \vee (x_{11} \wedge x_{NN}) \vee \\ \dots (x_{(N-1)(N-1)} \wedge x_{NN})) = \text{True}$$

- La restricción anterior corresponde a una de las diagonales centrales. Para tener un modelo completo habría que agregar restricciones similares para todas las diagonales restantes.

De manera similar es posible modelar otros *CSP/CSOP* como problemas *SAT* para luego resolverlos por medio de *solvers* especializados en operaciones booleanas, por ejemplo, *Z3* [15].

2.3. SMT

Satisfiability Modulo Theories (SMT) [16] es un framework que busca resolver el problema de verificar la satisfacibilidad de una fórmula lógica de primer orden con respecto a la unión de una o más lógicas. *SMT* combina problemas de satisfacción booleana (*SAT*) con los distintos dominios dentro del problema de decisión general abordado, lo que implica una alta complejidad computacional y de modelamiento para poder combinar los distintos algoritmos especializados para cada tarea del problema [17].

SMT separa los dominios a nivel general en un conjunto de *teorías*, de tal manera que el dominio para representar los números enteros quede separado del dominio de los números reales y de los números binarios, entre otros. Luego, a partir de estas teorías

surgen las denominadas *lógicas SMT*, las cuales se componen por la combinación entre una o más *teorías SMT*, en base a lo descrito en la versión estándar de la *SMT-LIB 2.6* [18]. A diferencia de *SAT* que solo utiliza álgebra booleana para determinar si un problema es *sat* o *unsat*, *SMT* introduce la combinación entre álgebra booleana y otras teorías de primer orden para resolver el mismo problema, permitiendo mayor expresión y flexibilidad en el modelamiento matemático de problemas.

Si definimos una fórmula lógica como φ , el problema que define a *SMT* es el de validar si φ es *sat* o *unsat* en el contexto definido por una teoría $\mathcal{T}$ que restringe la interpretación de los símbolos usados en φ , es decir, *SMT* valida si existe un modelo de $\mathcal{T}$ con el que φ sea *sat* (o retorne verdadero).

Las teorías que componen la *SMT-LIB 2.6* son las siguientes:

- *ArraysEx*: Arreglos funcionales con extensionalidad.
- *FixedSizeBitVectors*: Vectores de bits de tamaño arbitrario.
- *Boolean* o *SAT (core)*: Teoría central de *SMT*, define los operadores booleanos básicos.
- *FloatingPoint*: Números punto flotante.
- *Ints*: Números enteros.
- *Reals*: Números reales.
- *Reals_Ints*: Números reales y enteros.
- *Strings*: Cadenas de caracteres *unicode* y expresiones regulares.

A partir de las teorías anteriores es posible componer las *lógicas SMT* mediante la combinación de una o más teorías. En este documento nos centramos en las siguientes 4 lógicas de las mostradas en la Figura 2.2:

- *Linear Integer Arithmetic (LIA)*:
LIA surge como una extensión de la teoría de números enteros (*Ints*) y de la teoría de booleanos (*Core*). Permite representar combinaciones booleanas de inecuaciones entre polinomios lineales sobre variables con dominio en los números enteros.
- *Linear Real Arithmetic (LRA)*:
LRA surge como una extensión de la teoría de números reales (*Reals*) y de la teoría de booleanos (*Core*). Permite representar combinaciones booleanas de inecuaciones entre polinomios lineales sobre variables con dominio en los números reales.

SMT define internamente el siguiente ciclo de búsqueda para determinar si un modelo es *sat* o *unsat* [19, 20]:

1. Se define el modelo original, $\mathcal{P}$, con una o más lógicas de primer orden.
2. *SMT* transforma el modelo entregado en una serie de predicados y operadores booleanos para construir un nuevo modelo simplificado de tipo *SAT*, $\mathcal{P}_S$, y revisa las condiciones para que éste sea *sat*, es decir, $\mathcal{P}_S \rightarrow True$. Si se determina que el modelo $\mathcal{P}_S$ es *unsat*, entonces se concluye que el modelo original $\mathcal{P}$ también es *unsat* y el ciclo termina.
3. Luego construye un nuevo modelo, $\mathcal{P}_L$, con las lógicas del modelo original, $\mathcal{P}$, usando los predicados resultantes en $\mathcal{P}_S$, pero planteando como hipótesis que este nuevo modelo es *unsat*, es decir, $\mathcal{P}_L \rightarrow False$.
4. Luego construye un nuevo modelo *SAT* a partir de $\neg \mathcal{P}_L \rightarrow True$ denominado $\mathcal{L}_S$ y plantea como hipótesis que $\mathcal{P}_S \wedge \mathcal{L}_S \rightarrow True$.
5. Si la hipótesis $\mathcal{P}_S \wedge \mathcal{L}_S \rightarrow True$ es *unsat*, entonces se concluye por *prueba por contradicción* que el problema original $\mathcal{P}$ también es *unsat*. En caso contrario, se vuelve al paso 2 y el ciclo continúa su ejecución hasta concluir que el modelo es *unsat* hasta alcanzar convergencia, la cual está asegurada dado que la cantidad de átomos presentes en la fórmula es finita, y por consecuencia, el número de modelos intermedios también lo es [19].

2.3.1. Aplicaciones

- **Problemas de satisfacción de restricciones (y optimización):**

El uso de *SMT* como alternativa a *SAT* para modelar problemas de optimización combinatoria es cada vez mayor gracias a la flexibilidad que ofrece poder modelar usando teorías que se apeguen al contexto de un determinado problema en vez de tener que modelar directamente sobre álgebra lineal de booleanos, razón por la cual aplicaciones de *SMT* y mejoras para los *solvers* que lo emplean se encuentran en constante auge [1, 21, 22, 23, 24].

- **Hardware/Software checking:**

Haciendo uso de fórmulas lógicas se puede describir estados de un programa o dispositivo y transformaciones entre estos estados, proceso que resulta clave en herramientas que analizan, verifican o realizan pruebas sobre programas o dispositivos físicos [19, 25, 26, 27, 28].

- **Automated Theorem Provers:**

Otro uso relevante de *SMT* se ve en la implementación de algoritmos para validar si es que una expresión matemática o lógica es formalmente válida o *sat*, como es

el caso de *Z3*, el cual puede manejar múltiples teorías y lógicas *SMT* para llevar a cabo estas validaciones [15, 29, 30, 31, 32].

2.4. Optimization Modulo Theories (OMT)

Como se ha visto hasta ahora, *SMT* se especializa en verificar la satisfacibilidad de fórmulas y modelos matemáticos, por lo que es una opción sólida para abordar problemas de satisfacción de restricciones.

OMT [1] nace con la finalidad de extender el *framework* de *SMT* y agregarle una herramienta que le permita también encontrar un valor óptimo acorde a una función objetivo. Esta primera implementación de *OMT* se enfoca en *LIA*, por lo que se consideraron dos maneras para lograrlo:

- La primera consiste en utilizar el mejor *solver* comercial disponible en esa fecha (*CPLEX*) y usarlo como optimizador de programación lineal externo.
- La segunda consiste en la integración de una técnica similar a la implementada en el *T-solver* [33] y modificar el *framework* para determinar satisfacibilidad de *SMT* para que incluya un algoritmo de búsqueda de óptimo lineal. Se considera *T-solver* dado que desde el 2006 hasta el 2011 (fecha de publicación de *OMT*) ha sido el *solver* utilizado por todos los ganadores en la categoría *LIA/LRA* de la *SMT-COMP* [34], evento en el que se compete por resolver problemas de satisfacción de restricciones (y optimización) usando *SMT*.

Hoy en día, herramientas basadas en *OMT* permiten optimizar problemas modelados a partir de casi cualquier combinación de teorías presentes en la *SMT-LIB* [35, 36, 37], siendo *z3* [15, 38], *OptiMathSAT* [39] y *CVC5* [40] algunos de los *solvers SMT/OMT* modernos más relevantes en el área.

En este documento se utiliza *Z3* [15], el *solver SMT* de *Microsoft* para resolver los problemas modelados en el Capítulo 3. Inicialmente, *Z3* se enfocaba solo en *SMT* como un *Theorem Prover*, pero con el tiempo incorporó el *solver OMT vZ* [38], integrando con ello la colección de *solvers MaxSAT* para abordar restricciones blandas, y módulos *OptSMT* para optimizar en base a funciones objetivo con aritmética lineal [41].

2.5. Definición y formalización del problema

La resolución de Problemas de satisfacción de restricciones y/o optimización (CSP/C-SOP) usando *SMT* se vuelve cada más frecuente en distintas áreas de la ingeniería, lo que ha impulsado múltiples estudios para mejorar el desempeño de los *solvers* y de las estrategias de búsqueda que hay por debajo de estos [42, 43, 44, 45, 46, 47, 48, 39, 49,

50, 51]. Sin embargo, son pocos los estudios que han abordado la comparación entre los codificaciones (*encodings*) o lógicas que dispone la *SMT-LIB* para modelar este tipo de problemas.

Generalmente se usa la lógica que más se apegue al contexto de un problema o que cumpla con una lista de requisitos técnicos como es el caso de algunos sistemas *embedded* [5], por lo que no se ha dado la oportunidad de estudiar a fondo las diferencias de desempeño que puede producir el uso de una determinada lógica *SMT* por sobre otra.

Ewan et al. [6] mencionó este fenómeno, mostrando que hay *CSOP* para los que se producen distintas soluciones al usar distintas lógicas *SMT* como base del modelamiento matemático, obteniendo como resultado que al abordar el mismo problema usando lógica de *bit-vectors* es posible obtener mejores soluciones en comparación a modelos basados en otras lógicas *SMT*. También se tiene el estudio de *Miquel et al.* [21], en donde sugiere que a medida que aumenta la cantidad de componentes booleanas dentro un determinado problema, mejora el desempeño general de *SMT* al acercarlo más a un modelo de tipo *SAT* puro, reduciendo la cantidad de tareas internas que debe resolver el *solver* para modelos *SMT*.

La falta de estudios particularmente enfocados en la comparación de lógicas como parámetros del *solver* más que como una decisión de diseño abre la posibilidad a que existan diferencias estadísticamente significativas y cuantificables en la resolución de problemas, en donde podrían existir patrones que asocien un determinado tipo de problema (*scheduling*, *planning*, *set covering*, *set partitioning*, *set packing*, entre otros), o incluso un determinado tipo de restricción, a una lógica *SMT* que permita optimizar la búsqueda de soluciones, siendo esto finalmente lo que motiva el estudio llevado a cabo en este documento.

De manera formal, el problema que se busca abordar es el siguiente:

Sea $\mathcal{L}$ como un conjunto de lógicas *SMT*, tal que $(l, m \in \mathcal{L})$, y Θ como un conjunto de *CSP/CSOP*, tal que $(\theta \in \Theta)$, queremos ver si se cumple que $model(l, \theta) > model(m, \theta), l \neq m, \forall \theta \in \Theta$.

Si la formalización anterior se cumple significaría que efectivamente el tipo de lógica *SMT* elegida impacta directamente en la resolución de problemas, lo que afectaría directamente al modelamiento de todos los *CSP/CSOP* que se basen en lo definido por la *SMT-LIB*.

Propuesta de solución

Para abordar el problema de decidir que combinación de teorías o que lógica utilizar para modelar un determinado *CSOP* se propone resolver un conjunto de instancias para problemas conocidos de satisfacción de restricciones mediante múltiples modelamientos matemáticos, tal que cada modelo se construya utilizando una lógica *SMT*, lo que permite obtener una comparación directa en el comportamiento del *solver* utilizado frente a cada una de las lógicas *SMT* elegidas al abordar problemas con distintas áreas de especialidad, como por ejemplo *Scheduling* o *Planning*.

Para ello, primero se define un conjunto de problemas de satisfacción de restricciones y optimización, luego se define un conjunto de instancias con distinta complejidad para cada problema, seguido por el modelamiento matemático de cada problema utilizando la lógica *SMT LIA*, es decir, todos los dominios de los problemas estarán acotados por un conjunto de números enteros. Después de modelar los problemas usando *LIA* se define un procedimiento para transformar cada modelo en uno nuevo, pero usando las otras tres lógicas descritas en la Sección 2.3. Por último, se resuelven todos los problemas de optimización usando *OMT* por medio del *solver Z3* para obtener los resultados registrados en el Capítulo 4.

3.1. Problemas a analizar

Los problemas de decisión considerados para llevar a cabo la comparación descrita son los siguientes:

- *N-queens*

N-queens es el problema 054 de la *CSPLib* [8] y tiene un nivel de complejidad *NP*-completo [52]. Consiste en posicionar *N* reinas del mismo color en un tablero

de ajedrez de tamaño $N \times N$ de tal manera que ninguna reina pueda atacar o ser atacada por otra reina.

- ***Balanced Academic Curriculum Problem (BACP)***

BACP es el problema 030 de la *CSPLib* [53] y también tiene un nivel de complejidad *NP-completo* [54], dado que la función objetivo es polinomialmente computable (*NP*) y dado que es posible reducir una instancia del problema *3-partition* al problema de decisión planteado en *BACP* (completitud) [55].

BACP consiste en distribuir cursos en periodos académicos satisfaciendo pre-requisitos entre cursos y balanceando los créditos de los cursos entre cada periodo, de manera que se pueda evitar terminar con periodos con muchos o muy pocos créditos en comparación al resto.

- ***Nurse Scheduling Problem (NSP)***

NSP es un problema que se ha estudiado desde antes de 1969 [56] y consiste en asignar *nurses* a turnos de trabajo por día, tal que la calidad de la asignación sea la mayor posible mientras se cumple con las restricciones duras y blandas del problema.

Dada la similitud de *NSP* con *Machine Scheduling*, se asume o espera que *NSP* tenga un nivel de complejidad *NP-hard* [57].

- ***Travelling Salesman Problem (TSP)***

TSP es un problema que se define de la siguiente manera [58]: dada una lista de N ciudades y las distancias entre cualquier combinación de 2 ciudades, se debe calcular una ruta que pase por cada ciudad una sola vez salvo por la ciudad de origen, la cual debe ser visitada al inicio y al final de la ruta. La función objetivo de este problema implica minimizar la distancia total recorrida.

TSP tiene una complejidad *NP-completo* cuando es abordado como un problema de decisión dado que se puede obtener como una reducción del *Hamiltonian Cycle Problem* [59], pero cuando se aborda como un problema de optimización la complejidad sube a *NP-hard* dado que no es posible concluir que el ciclo Hamiltoniano encontrado es el más corto en tiempo polinomial, razón por la que se vuelve necesario enumerar todas las soluciones posibles y en base a ello decidir cuál es el ciclo más corto.

- ***Unbounded Knapsack Problem (UKP)***

UKP es una variación del problema 133, *Knapsack Problem*, propuesto en la *CSPLib* [60]. El problema original se define de la siguiente manera: dada una mochila con capacidad C y una lista de N objetos con pesos W_n y utilidad V_n , se debe buscar una combinación de objetos, sin repetición, que maximice la utilidad total sin superar la capacidad de la mochila.

Al igual que el *Knapsack Problem*, se considera a *UKP* como uno de los problemas *NP-completos* más simples dado que solo tienen una restricción, a diferencia de otros problemas conocidos [61].

3.1.1. Instancias

Las instancias utilizadas fueron elegidas de manera que cada problema cuente con instancias de distinta complejidad para poder comparar el comportamiento de los *solvers* especializados en cada lógica a medida que va aumentando la complejidad del modelo matemático.

■ *N-queens*

Para este problema, cada instancia se define con el tamaño del tablero y la cantidad de reinas a posicionar en base al valor de N . Si $N = 4$ significa que el tablero será de tamaño 4×4 y se deberán posicionar 4 reinas.

En base a lo anterior, se determinó el siguiente conjunto de números para N :

$$N = \{4, 8, 12, 16, 20, 50, 100, 120, 140, 160, 180, 200\}$$

Formando en total 12 instancias únicas para este problema.

■ *Balanced Academic Curriculum Problem (BACP)*

Para este problema se eligieron las tres instancias publicadas en la página de la *CSPLib* [62]. Las instancias se construyen a partir de la malla curricular de tres carreras de informática de la Universidad Técnica Federico Santa María y contienen la siguiente información:

- **Currículum académico (lista de *strings*):**

Conjunto de cursos o asignaturas pertenecientes al currículum de la carrera y la relación de precedencia entre entre éstos.

- **Número de periodos (número entero):**

La cantidad de periodos máxima en la que se deberán asignar todos los cursos.

- **Carga académica (lista de números enteros):**

Cada curso tiene asignado una cantidad de créditos acorde al esfuerzo académico que requiere para completarlo exitosamente.

- **Pre-requisitos (lista de pares de *strings*):**

Conjunto con los pares (curso, pre-requisito) para definir el orden de precedencia entre cursos. Un curso puede ser pre-requisito de cero o más cursos, y un curso puede tener cero o más cursos como pre-requisitos.

- **Carga académica mínima (número entero):**

Cantidad mínima de créditos que se deben asignar a un estudiante por periodo académico para que sea considerado estudiante de tiempo completo.

- **Carga académica máxima (número entero):**

Cantidad máxima de créditos que se deben asignar a un estudiante por periodo académico para evitar sobrecarga.

- **Cantidad mínima de cursos (número entero):**

Cantidad mínima de cursos que se deben asignar a un estudiante por periodo académico para que sea considerado estudiante de tiempo completo.

- **Cantidad máxima de cursos (número entero):**

Cantidad máxima de cursos que se deben asignar a un estudiante por periodo académico para evitar sobrecarga.

Las 3 instancias elegidas se diferencian por la cantidad de periodos (8, 10 y 12, respectivamente) y por los cursos (y sus relaciones) de cada currículum.

- ***Nurse Scheduling Problem (NSP)***

Para este problema se eligieron 6 de las instancias publicadas en la *NSP-LIB* [63]. Las instancias se generan a partir de la combinatoria producto de asignarle valores duros a las restricciones blandas de preferencia y cobertura, lo que le permite a la *NSP-LIB* contar con 4 conjuntos de 7290 instancias para las instancias de tipo *diversas* y 2 conjuntos de 960 instancias para las instancias de tipo *realistas*.

Cada instancia generada contiene la siguiente información:

- **Tamaño del problema:**

Definido por la cantidad N de *nurses*, la cantidad D de días y la cantidad S de turnos por día. Éstos 3 valores se representan como números enteros.

- **Matriz de cobertura:**

Matriz de números enteros con D filas y S columnas. Los valores de esta matriz indican la cantidad de *nurses* que se necesitan para un determinado turno (representado por el número de la columna seleccionada) en un determinado día (representado por el número de la fila seleccionada).

- **Matriz de preferencias:**

Matriz de números enteros con N filas y $D \times S$ columnas. Los valores de esta matriz indican la preferencia que tiene un determinado *nurse* (representado por el número de la fila seleccionada) para trabajar en cada turno de cada día.

Además, la *NSP-LIB* propone 16 casos específicos de estudio con los que se produce información adicional para el modelamiento de restricciones:

- **Cantidad mínima de días asignados a cada *nurse*:**

Cantidad mínima de días que se deben asignar a un *nurse* por jornada de 7 o 28 días (dependiendo del tipo de instancia) para que cumpla con los requisitos de contrato.

- **Cantidad máxima de días asignados a cada *nurse*:**

Cantidad máxima de días que se deben asignar a un *nurse* por jornada de 7 o 28 días (dependiendo del tipo de instancia) para que cumpla con los requisitos de contrato.

- **Cantidad mínima de días consecutivos que se deben asignar a un *nurse*:**

Cada vez que se asigne un *nurse* a un determinado día de trabajo, éste deberá continuar trabajando los días continuos hasta al menos completar la cantidad mínima de días consecutivos.

- **Cantidad máxima de días consecutivos que se pueden asignar a un *nurse*:**

Cada vez que se asigne un *nurse* a un determinado día de trabajo, éste podrá continuar trabajando los días continuos mientras no supere la cantidad máxima de días consecutivos.

- **Cantidad mínima de días consecutivos que se debe asignar el mismo turno a un *nurse*:**

Cada vez que se asigne un *nurse* a un determinado turno de trabajo, éste deberá continuar trabajando el mismo turno durante los días continuos hasta al menos completar la cantidad mínima de días consecutivos por turno indicada.

- **Cantidad máxima de días consecutivos que se puede asignar el mismo turno a un *nurse*:**

Cada vez que se asigne un *nurse* a un determinado turno de trabajo, éste podrá continuar trabajando el mismo turno durante los días continuos mientras no supere la cantidad máxima de días consecutivos por turno indicada.

- **Cantidad mínima de veces que se debe asignar un determinado turno a cada *nurse*:**

A cada *nurse* se le deberá asignar cada turno de trabajo una cantidad mínima de veces durante la jornada de 7 o 28 días.

- **Cantidad máxima de veces que se puede asignar un determinado turno a cada *nurse*:**

A cada *nurse* se le podrá asignar cada turno de trabajo una cantidad máxima de veces durante la jornada de 7 o 28 días.

Adicionalmente, la *NSP-LIB* considera que el último turno de cada día es de descanso y por lo tanto no deberá tener ningún *nurse* asignado.

El objetivo para estas instancias es satisfacer las restricciones descritas en la información de las instancias y maximizar la preferencia asignando *nurses* a turnos con mayor valor en la matriz de preferencias.

De las 6 instancias elegidas, 4 corresponden al conjunto de instancias ***diversas***, las que se caracterizan por tener 4 turnos por día y una jornada de 7 días. Estas 4 instancias se diferencian entre sí por tener 25, 50, 75 y 100 *nurses*, respectivamente. Las 2 instancias restantes pertenecen al conjunto de instancias ***realistas*** y se caracterizan por tener 4 turnos por día, pero con jornadas de 28 días. Estas 2 instancias tienen 30 y 60 *nurses*, respectivamente.

Por completitud, los nombres de las instancias trabajadas son los siguientes:

- Instancias *diversas* (gen 8):
25_1.nsp, 50_1.nsp, 75_1.nsp, 100_1.nsp
- Instancias *realistas* (gen 16):
30_1.nsp, 60_1.nsp

Para los tipos de instancias trabajadas se eligieron los casos específicos *gen 8* y *gen 16* dado que incluyen mayor variedad de restricciones presentes en el segundo grupo de información de las instancias.

■ *Travelling Salesman Problem (TSP)*

Para este problema se eligieron 5 instancias [64] y 3 de éstas pertenecen a la *TSP-LIB* [65].

La principal diferencia entre estas 5 instancias es la cantidad de ciudades que componen la red de posibles rutas:

- Instancia “*five*”: 5 ciudades, **largo mínimo del tour:** 19
- Instancia “*p01*”: 15 ciudades, **largo mínimo del tour:** 291
- Instancia “*gr17*”: 17 ciudades, **largo mínimo del tour:** 2085
- Instancia “*fri26*”: 26 ciudades, **largo mínimo del tour:** 937
- Instancia “*att48*”: 48 ciudades, **largo mínimo del tour:** 33523

Cada instancia se representa como una matriz de adyacencia, de tal manera que cada casilla representa la distancia que hay entre la ciudad indicada por el número de la fila y la ciudad indicada por el número de la columna.

Las instancias elegidas son simétricas, es decir, la distancia entre una ciudad x y una ciudad y es la misma en ambas direcciones.

El objetivo para este problema es generar el *tour* o ruta que recorra todas las ciudades, terminando en la ciudad inicial, que minimice la distancia total recorrida.

■ *Unbounded Knapsack Problem (UKP)*

Para este problema se usó un generador de instancias artificiales [66] para generar 8 instancias únicas en las que varía el número de objetos disponibles y la capacidad de la mochila:

- Instancia “*n10-0-c7370*”: $N = 10$ objetos, capacidad $C = 7370$
- Instancia “*n20-1-c17050*”: $N = 20$ objetos, capacidad $C = 17050$
- Instancia “*n30-0-c27795*”: $N = 30$ objetos, capacidad $C = 27795$
- Instancia “*n40-3-c39588*”: $N = 40$ objetos, capacidad $C = 39588$
- Instancia “*n50-4-c41934*”: $N = 50$ objetos, capacidad $C = 41934$
- Instancia “*n60-2-c57601*”: $N = 60$ objetos, capacidad $C = 57601$

- **Instancia “n70-3-c68135”:** $N = 70$ objetos, capacidad $C = 68135$
- **Instancia “n80-0-c67603”:** $N = 80$ objetos, capacidad $C = 67603$
- El “-0-” en el nombre de la instancia significa que se usó la *seed* 731232778 al momento de generarla.
- El “-1-” en el nombre de la instancia significa que se usó la *seed* 594914841 al momento de generarla.
- El “-2-” en el nombre de la instancia significa que se usó la *seed* 324673955 al momento de generarla.
- El “-3-” en el nombre de la instancia significa que se usó la *seed* 300958364 al momento de generarla.
- El “-4-” en el nombre de la instancia significa que se usó la *seed* 182130978 al momento de generarla.

Cada instancia contiene la siguiente información:

- Número de objetos N y capacidad C de la mochila
- Lista de números enteros con la utilidad v de cada objeto
- Lista de números enteros con el peso w de cada objeto

El objetivo de este problema es agregar 0 o más veces cada objeto a la mochila de manera que se maximice la utilidad de los objetos agregados sin que se sobrepase la capacidad de la mochila.

3.2. Modelamiento inicial de problemas usando LIA

Como condición para la comparación de modelos, se consideró que los modelos de cada problema usando cada una de las 4 lógicas *SMT* descritas en la Sección 2.3 deben ser lo más similares posibles en cuanto a cantidad y tipo de restricciones para así poder comparar todos los modelos en condiciones similares. Es por ello que se tomó la decisión de hacer un modelamiento inicial usando únicamente álgebra lineal de enteros (LIA) y luego traducir o transformar este modelo a las otras 3 lógicas sin alterar las decisiones de diseño tomadas en el modelo inicial.

3.2.1. *N-queens*

Una forma de modelar este problema es siguiendo el procedimiento visto en la Sección 2.1.1, sin embargo, es posible lograr una mejor representación del problema (o al menos una que requiere de un menor número de variables) usando el siguiente modelo:

■ **Parámetros:**

N : Número de reinas que se deben posicionar en un tablero de tamaño $N \times N$, tal que $N \in \mathbb{Z}^+$

■ **Variables:**

X_c : Número de la fila en la columna c donde se posiciona una reina, $X_c \in \mathbb{Z}, \forall c \in \{1, \dots, N\}$

■ **Función objetivo:**

Que ninguna reina en el tablero pueda ser atacada, o en otras palabras, que se cumplan todas las restricciones del problema

■ **Restricciones del problema:**

1. El dominio de X_c está restringido entre 1 y N (cantidad de filas):

$$X_c \geq 1,$$

$$X_c \leq N,$$

$$\forall c \in \{1, \dots, N\}$$

2. Debe haber a lo mas una reina por fila:

$$X_{c_i} \neq X_{c_j}$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_i + 1, \dots, N\}$$

3. Debe haber a lo mas una reina por diagonal

$$X_{c_j} \neq X_{c_i} + (c_j - c_i)$$

$$X_{c_i} \neq X_{c_j} + (c_j - c_i)$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_i + 1, \dots, N\}$$

3.2.2. *Balanced Academic Curriculum Problem (BACP)*

El modelamiento de este problema se construyó a partir del modelo de programación lineal entera presentado en *Castro et al.* [62]:

■ **Parámetros:**

m : Número de cursos, $m \in \mathbb{Z}^+$

n : Número de periodos académicos, $n \in \mathbb{Z}^+$

α_i : Número de créditos del curso i , $\alpha_i \in \mathbb{Z}^+$, $\forall i \in \mathbb{Z}^+ : i \in \{1, \dots, m\}$

β : Carga académica **mínima** permitida por periodo, $\beta \in \mathbb{Z}^+$

γ : Carga académica **máxima** permitida por periodo, $\gamma \in \mathbb{Z}^+$

δ : Cantidad de cursos **mínima** por periodo, $\delta \in \mathbb{Z}^+$

ϵ : Cantidad de cursos **máxima** por periodo, $\epsilon \in \mathbb{Z}^+$

■ **Variables:**

$$X_{i,j} = \begin{cases} 1 & \text{si el curso } i \text{ es asignado al periodo } j, \\ & X_{i,j} \in \mathbb{Z}, \forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\} \\ 0 & \text{en caso contrario} \end{cases}$$

c_j : carga académica del periodo j , $c_j \in \mathbb{Z}, \forall j \in \{1, \dots, n\}$

C : Mayor carga académica registrada a lo largo del currículum, $C \in \mathbb{Z}$

■ **Función objetivo:**

Minimizar la carga académica a lo largo de los periodos, lo que se puede representar como: minimizar la mayor carga académica registrada a lo largo del currículum:

Minimizar C

■ **Restricciones del problema:**

1. El dominio de $X_{i,j}$ está restringido entre 0 y 1:

$$X_{i,j} \leq 1,$$

$$X_{i,j} \geq 0,$$

$$\forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\}$$

2. La mayor carga académica C se define como:

$$C = \text{Max}\{c_1, \dots, c_n\}$$

Lo que se puede forzar usando la siguiente restricción lineal en conjunto con la función objetivo:

$$C \geq c_j, \forall j \in \{1, \dots, n\}$$

3. La carga académica para el periodo j se define como:

$$c_j = \sum_{i=1}^m \alpha_i \times X_{i,j}, \forall j \in \{1, \dots, n\}$$

4. Cada curso i debe ser asignados a un periodo j :

$$\sum_{j=1}^n X_{i,j} = 1, \forall i \in \{1, \dots, m\}$$

5. Curso b tiene como pre-requisito al curso a , por lo tanto el curso b no se puede asignar a un periodo anterior al asignado para el curso a :

$$X_{b,j} \leq \sum_{r=1}^{j-1} X_{a,r} = 1, \forall j \in \{2, \dots, n\}, \forall a \in \{1, \dots, m-1\}, \forall b \in \{a, \dots, m\}$$

6. La carga académica de cada periodo j debe ser mayor o igual a la carga mínima permitida:

$$c_j \geq \beta, \forall j \in \{1, \dots, n\}$$

7. La carga académica de cada periodo j debe ser menor o igual a la carga máxima permitida:

$$c_j \leq \gamma, \forall j \in \{1, \dots, n\}$$

8. El número de cursos del periodo j debe ser mayor o igual a la cantidad mínima permitida:

$$\sum_{i=1}^m X_{i,j} \geq \delta, \forall j \in \{1, \dots, n\}$$

9. El número de cursos del periodo j debe ser menor o igual que la cantidad máxima permitida:

$$\sum_{i=1}^m X_{i,j} \leq \epsilon, \forall j \in \{1, \dots, n\}$$

3.2.3. *Nurse Scheduling Problem (NSP)*

El modelo para este problema se construyó a partir de las restricciones mencionadas en el *paper* de la *NSP-LIB* [63], incluyendo las restricciones para los casos específicos mencionados en la Sección 3.1.1:

■ **Parámetros:**

N : Número de *nurses*, $N \in \mathbb{Z}^+$

D : Número de días que definen la jornada de trabajo, $D \in \mathbb{Z}^+$

S : Número de turnos por día (el último no turno se considera como turno de trabajo, es decir, si $S = 4$, el día contará con $S - 1 = 3$ turnos laborales), $S \in \mathbb{Z}^+$

$C_{d,s}$: Matriz de cobertura, indica la cantidad de *nurses* requerida para el día d y el turno s , $C_{d,s} \in \mathbb{Z}_0^+, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$P_{n,d,s}$: Matriz de preferencias, indica la preferencia que tiene un *nurse* n para trabajar el día d en el turno s , $P_{n,d,s} \in \mathbb{Z}_0^+, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

min_wd : Cantidad mínima de días en que a cada *nurse* se le debe asignar un turno laboral, $min_wd \in \mathbb{Z}_0^+$

max_wd : Cantidad máxima de días en que a cada *nurse* se le debe asignar un turno laboral, $max_wd \in \mathbb{Z}_0^+$

min_cd : Cantidad mínima de días de trabajo consecutivos, $min_cd \in \mathbb{Z}_0^+$

max_cd : Cantidad máxima de días de trabajo consecutivos, $max_cd \in \mathbb{Z}_0^+$

min_cs_s : Número entero que indica la cantidad mínima de días consecutivos que un *nurse* deberá trabajar en el turno s , en caso de que se le sea asignado, $min_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

max_cs_s : Número entero que indica la cantidad máxima de días consecutivos que un *nurse* podrá trabajar en el turno s , $max_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

min_s_s : Número entero que indica la cantidad mínima de veces que se debe asignar el turno s a cada *nurse*, $min_s_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

max_s_s : Número entero que indica la cantidad máxima de veces que se puede asignar el turno s a cada *nurse*, $max_s_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

■ **Variables:**

$$X_{n,d,s} = \begin{cases} 1 & \text{si el } nurse \ n \text{ es asignado al día } d \text{ en el turno } s, \\ & X_{n,d,s} \in \mathbb{Z}, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\} \\ 0 & \text{en caso contrario} \end{cases}$$

■ **Función objetivo:**

Maximizar $H = \sum_{n=1}^N \sum_{d=1}^D \sum_{s=1}^S X_{n,d,s} \times P_{n,d,s}$ (*Happiness*, determinado por los turnos asignados a cada *nurse* y su preferencia sobre estos), $\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

■ **Restricciones del problema:**

1. El dominio de $X_{n,d,s}$ está restringido entre 1 y 0:

$$X_{n,d,s} \geq 0,$$

$$X_{n,d,s} \leq 1,$$

$$\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$$

2. Cada *nurse* n debe tener asignado un turno s cada día:

$$\sum_{s=1}^S X_{n,d,s} = 1, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}$$

3. Cada turno laboral s en cada día d deberá tener al menos la cantidad $C_{d,s}$ de *nurses* indicada en la Matriz de cobertura:

$$\sum_{n=1}^N X_{n,d,s} \geq C_{d,s}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S-1\}$$

4. A cada nurse se le debe asignar al menos min_wd y a lo más max_wd días de trabajo (sin contar el turno libre):

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \geq min_wd, \forall n \in \{1, \dots, N\}$$

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \leq max_wd, \forall n \in \{1, \dots, N\}$$

5. Cada *nurse* debe tener asignado un determinado turno s al menos min_s_s y a lo más max_s_s veces:

$$\sum_{d=1}^D X_{n,d,s} \geq min_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

$$\sum_{d=1}^D X_{n,d,s} \leq max_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

6. Cada *nurse* debe trabajar al menos min_cd y a lo más max_cd días consecutivos

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} \sum_{s=1}^{S-1} X_{n,j-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i+k-1,s} \leq k-2,$$

$$\forall n \in \{1, \dots, N\}, \forall k \in \{\min\{2, min_cd\}, \dots, min_cd+1\}, \forall i \in \{1, \dots, D+2-k\}$$

$$\sum_{d=j}^{\min\{D,j+max_cd+1\}} \sum_{s=1}^{S-1} X_{n,d,s} \leq max_cd,$$

$$\forall n \in \{1, \dots, N\}, \forall j \in \{1, \dots, D - max_cd\}$$

7. Cada *nurse* debe repetir el mismo turno de trabajo s al menos min_cs_s y a lo más max_cs_s veces, en caso de que el turno s le sea asignado

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} X_{n,j-1,s} - 2 \times X_{n,i-1,s} - 2 \times X_{n,i+k-1,s} \leq k-2,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S-1\},$$

$$\forall k \in \{\min\{2, \min_cs_s\}, \dots, \min_cs_s + 1\}, \forall i \in \{1, \dots, D + 2 - k\}$$

$$\sum_{d=j}^{\min\{D, j+\max_cs_s+1\}} X_{n,d,s} \leq \max_cs_s,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S - 1\}, \forall j \in \{1, \dots, D - \max_cd\}$$

3.2.4. Travelling Salesman Problem (TSP)

El modelo para este problema se construyó a partir del modelo de programación lineal entera presentado en *Miller et al.* [67]:

- **Parámetros:**

n : Número de nodos, $n \in \mathbb{Z}^+$

$D_{i,j}$: Distancia entre el nodo i y el nodo j , $D_{i,j} \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$

- **Variables:**

$$X_{i,j} = \begin{cases} 1 & \text{si el nodo } j \text{ es visitado desde el nodo } i, \\ & X_{i,j} \in \mathbb{Z}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\} \\ 0 & \text{en caso contrario} \end{cases}$$

u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}, \forall i \in \{1, \dots, n\}$

- **Función objetivo:**

Minimizar distancia total recorrida $d = \sum_{i=1}^n \sum_{j=1}^n X_{i,j} \times D_{i,j}$

- **Restricciones del problema:**

1. El dominio de $X_{i,j}$ está restringido entre 1 y 0:

$$X_{i,j} \geq 0,$$

$$X_{i,j} \leq 1,$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

2. El dominio de u_i está restringido entre 1 y n :

$$u_i \geq 1,$$

$$u_i \leq n,$$

$$\forall i \in \{1, \dots, n\}$$

3. Cada nodo i debe ser un punto de partida una sola vez:

$$\sum_{j=1}^n X_{i,j} = 1, \forall i \in \{1, \dots, n\}$$

4. Cada nodo j debe ser punto de llegada una sola vez:

$$\sum_{i=1}^n X_{i,j} = 1, \forall j \in \{1, \dots, n\}$$

5. Eliminación de *sub-tours*:

$$u_i - u_j + n \times X_{i,j} \leq n - 1, \forall i, j \in \mathbb{Z}^+ : 2 \leq i \leq n, 1 \leq j \leq n$$

6. Eliminación de *tours* con ciudades repetidas:

$$u_i \neq u_j, \forall i, j \in \{1, \dots, n\} : i \neq j$$

7. Un nodo no se puede visitar a sí mismo (*self-loop*):

$$X_{i,i} = 0, \forall i \in \{1, \dots, n\}$$

3.2.5. *Unbounded Knapsack Problem (UKP)*

El modelamiento de este problema considera una mochila unidimensional con capacidad definida y objetos con peso positivo:

■ **Parámetros:**

n : Número entero con la cantidad total de objetos, $n \in \mathbb{Z}^+$

C : Número entero con la capacidad máxima de la mochila, $C \in \mathbb{Z}^+$

W_i : Número entero indicando el peso del objeto i , $W_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

V_i : Número entero indicando el valor del objeto i , $V_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

■ **Variables:**

X_i : cantidad de veces que se agrega el objeto i a la mochila, $X_i \in \mathbb{Z}, \forall i \in \{1, \dots, n\}$

■ **Función objetivo:**

Maximizar valor v en la mochila, $v = \sum_{i=1}^n V_i \times X_i$

■ **Restricciones del problema:**

1. Cada objeto puede ser ingresado cero o más veces:

$$X_i \geq 0, \forall i \in \{1, \dots, n\}$$

2. La suma de los pesos de los objetos ingresados no debe superar la capacidad máxima de la mochila:

$$\sum_{i=1}^n W_i \times X_i \leq C$$

3.3. Tipos de restricciones

De manera general, se extrae el siguiente resumen para categorizar todas las restricciones de los problemas vistos:

■ *N queens:*

- Variable numérica acotada inferior y superiormente por valor numérico
- Variable numérica distinta a otra variable numérica
- Variable numérica distinta a otra variable numérica más valor numérico

■ *BACP:*

- **Objetivo:** Minimizar sumatoria de variable pseudo booleana por valor numérico
- Variable pseudo booleana acotada inferior y superiormente por valor numérico
- Variable numérica acotada inferior y superiormente por valor numérico
- Variable numérica acotada inferiormente por variable numérica
- Sumatoria de variable pseudo booleana acotada a valor numérico
- Variable pseudo booleana acotada superiormente por sumatoria de variable pseudo booleana
- Sumatoria de variable pseudo booleana acotada inferiormente por valor numérico

■ *NSP:*

- **Objetivo:** Maximizar triple sumatoria variable pseudo booleana por valor numérico
- Variable pseudo booleana acotada inferior y superiormente por valor numérico
- Sumatoria de variable pseudo booleana acotada a valor numérico
- Sumatoria de variable pseudo booleana acotada inferiormente por valor numérico
- Doble sumatoria de variable pseudo booleana acotada inferior y superiormente por valor numérico
- Sumatoria de variable pseudo booleana acotada inferior y superiormente por valor numérico

■ *TSP:*

- **Objetivo:** Minimizar doble sumatoria variable pseudo booleana por valor numérico
 - Variable pseudo booleana acotada inferior y superiormente por valor numérico
 - Variable numérica acotada inferior y superiormente por valor numérico
 - Sumatoria de variable pseudo booleana acotada a valor numérico
 - Combinación lineal entre variable numérica, variable pseudo booleana y valor numérico acotada superiormente por valor numérico
 - Variable numérica distinta a otra variable numérica
 - Variable pseudo booleana acotada a valor numérico
- **UKP:**
- **Objetivo:** Maximizar sumatoria de variable numérica por valor numérico
 - Variable numérica acotada inferiormente por valor numérico
 - Sumatoria de variable numérica por valor numérico acotada superiormente por valor numérico

3.4. Traducción del modelamiento hacia otras lógicas *SMT*

El siguiente paso es transformar todos los modelos vistos en la Sección 3.2 utilizando otras lógicas *SMT*. Para ello se definen algunas reglas o pasos a seguir que permiten hacer estas transformaciones manteniendo a nivel general las mismas restricciones del modelo original usando *LIA*:

Dentro de cada sección con la transformación que corresponda, se incluye un ejemplo en el que se transforma el modelo descrito en la Sección 3.2.4 para *TSP*. Este modelo en particular se eligió dado que posee variables pseudo-booleanas (cuyos posibles valores son 1 o 0) y variables numéricas, por ende también tiene mayor variedad algebraica dentro de las restricciones y facilita la tarea de generalizar las transformaciones hacia los demás problemas.

Las transformaciones para los otros modelos de *N-queens*, *BACP*, *NSP* y *UKP* vistos en la Sección 3.2 están incluidas en el Anexo de este documento.

3.4.1. Transformación de *LIA* a *LRA*

Las transformaciones de modelos con dominio en los números enteros a modelos con dominio en los números reales son relativamente comunes y se conocen como *relajación*

de restricciones, sin embargo, si solo relajamos el dominio para que acepte todos los números reales tendremos el problema de que nuestro modelo entregará soluciones que, evaluadas desde el contexto original del problema, resultan infactibles en la práctica, como por ejemplo, agregar una fracción de un objeto a la mochila en el problema *UKP*.

Para lograr un cambio de lógica de números enteros a reales y seguir produciendo soluciones factibles se optó por cambiar el dominio de las variables a los números reales, pero ajustando las restricciones para restringir ese dominio a un conjunto finito (conocido) de número reales, es decir, el modelo resultante podrá usar todas las capacidades del *solver* para resolver problemas modelados con *LRA* usando un modelamiento similar a *LIA*.

En particular, las variables de tipo pseudo-booleo enteras seguirán siendo pseudo-booleas, solo que ahora los posibles valores serán 1,0 y 0,0 en vez de 1 y 0.

En el caso de las variables numéricas, estas se definirán como variables con dominio en los números reales y se agregará una nueva restricción para forzar a éstas variables a tomar un valor real definido dentro de un conjunto en vez de permitir el universo completo de los números reales.

Por ejemplo, consideremos la siguiente variable numérica del *TSP* modelado en la Sección 3.2.4:

- u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}$

La definición de esta variable al transformar el modelo a *LRA* es la siguiente:

- u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{R}_0^+, \forall i \in \{1, \dots, n\}$

Y la nueva restricción para limitar los valores que puede tomar u_i y evitar soluciones infactibles es la siguiente:

- $\bigvee_{j=1}^n u_i = j, \forall i \in \{1, \dots, n\}$

En el caso de la variable u_i se conoce que la cota superior es n gracias al contexto del problema, pero pueden haber casos en los que sea más difícil, o simplemente no viable acotar una variable usando este procedimiento.

La incorporación de operadores booleanos en las distintas lógicas *SMT* hace que sea posible construir un modelo usando la unión lógica para acotar el dominio.

A continuación se muestra el modelo completo usando *LRA* para el *TSP*:

■ **Parámetros:**

n : Número de nodos, $n \in \mathbb{Z}^+$

$D_{i,j}$: Distancia entre el nodo i y el nodo j , $D_{i,j} \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$

■ **Variables:**

$$X_{i,j} = \begin{cases} 1,0 & \text{si el nodo } j \text{ es visitado desde el nodo } i, \\ & X_{i,j} \in \mathbb{R}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\} \\ 0,0 & \text{en caso contrario} \end{cases}$$

u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{R}_0^+, \forall i \in \{1, \dots, n\}$

■ **Función objetivo:**

Minimizar distancia total recorrida $d = \sum_{i=1}^n \sum_{j=1}^n X_{i,j} \times D_{i,j}$

■ **Restricciones del problema:**

1. Acotamiento del dominio de $X_{i,j}$:

$$\bigvee_{k=0}^1 X_{i,j} = k, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

2. Acotamiento del dominio de u_i :

$$\bigvee_{j=1}^n u_i = j, \forall i \in \{1, \dots, n\}$$

3. El dominio de $X_{i,j}$ está restringido entre 1,0 y 0,0 (Notar que esta restricción se puede omitir en esta oportunidad dado que ya se encuentra considerada en el acotamiento de la restricción 1):

$$X_{i,j} \geq 0,0,$$

$$X_{i,j} \leq 1,0,$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

4. El dominio de u_i está restringido entre 1,0 y n (Notar que esta restricción se puede omitir en esta oportunidad dado que ya se encuentra considerada en el acotamiento de la restricción 2):

$$u_i \geq 1,0,$$

$$u_i \leq n,$$

$$\forall i \in \{1, \dots, n\}$$

5. Cada nodo i debe ser un punto de partida una sola vez:

$$\sum_{j=1}^n X_{i,j} = 1, 0, \forall i \in \{1, \dots, n\}$$

6. Cada nodo j debe ser punto de llegada una sola vez:

$$\sum_{i=1}^n X_{i,j} = 1, 0, \forall j \in \{1, \dots, n\}$$

7. Eliminación de *sub-tours*:

$$u_i - u_j + n \times X_{i,j} \leq n - 1, \forall i, j \in \mathbb{Z}^+ : 2 \leq i \leq n, 1 \leq j \leq n$$

8. Eliminación de *tours* con ciudades repetidas:

$$u_i \neq u_j, \forall i, j \in \{1, \dots, n\} : i \neq j$$

9. Un nodo no se puede visitar a sí mismo (*self-loop*):

$$X_{i,i} = 0, 0, \forall i \in \{1, \dots, n\}$$

3.4.2. Transformación de LIA a QF_BV

La transformación de modelos con dominio en los números reales a modelos con dominio en *bit-vectors* con tamaño predefinido es relativamente directa y depende en gran medida de que tan amplio es el dominio del problema original. Si este considera todos los enteros, entonces podría ser necesario usar complemento de dos para convertir todos los valores numéricos en vectores de bits. Si el dominio del problema original solo considera números iguales o mayores a cero, la codificación binaria es suficiente para lograr una buena transformación.

La transformación de variables pseudo-booleanas es directa. Estas originalmente solo toman como valor 1 y 0, por lo que se pueden representar con un vector de bits de tamaño 1.

La transformación de variables numéricas y de parámetros que formen parte de alguna combinación lineal se puede hacer directamente usando codificación binaria o complemento de dos, dependiendo de los signos que ésta pueda tomar. La cantidad mínima de bits para estas variables se puede obtener usando logaritmo en base 2 y la función techo. Por ejemplo, consideremos la variable u_i del problema *TSP* modelado en la Sección 3.2.4:

- u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}$

El dominio de u_i está en los enteros positivos incluyendo el cero, por lo que se puede transformar directamente usando codificación binaria. Para determinar la cantidad de bits que se le debe asignar a esta variable al momento de transformarla es necesario conocer el valor numérico máximo que ésta puede tomar. En este caso, ese valor corresponde a n , por lo que la cantidad mínima de bits es determinada con la siguiente ecuación:

$$nbits = \lceil \log_2(n) \rceil + 1$$

Por lo que la definición para la variable u_i en *QF_BV* usada en este documento corresponde a la siguiente:

- u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}_2^{nbits} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}$

Para este tipo de modelos usamos la notación $\mathbb{Z}_2^n$ para denotar una lista de n números enteros en base 2, donde cada número entero puede tomar los valores 0 o 1, tal que al concatenar los n elementos se forma el vector de bits correspondiente.

Un factor a tener en consideración al momento de hacer transformaciones de *LIA* a *QF_BV* es que la librería estándar de *SMT* [18] considera vectores de bits de tamaño fijo o estático, esto quiere decir que si una variable no se define desde el inicio con la cantidad de bits suficientes para almacenar algún posible valor correremos el riesgo de que el modelo sufra de *binary overflow* y se pierda información al momento de intentar resolverlo usando algún *solver*. Éstos suelen incluir métodos para flexibilizar operaciones entre *bit-vectors*, como es el caso de *Z3* que tiene incorporada una función para agregar temporalmente ceros a la izquierda de un vector de bits para poder contener valores producto de combinaciones de álgebra lineal entre las variables, al igual que operadores propios para distinguir entre *signed numbers* y *unsigned*.

A continuación se muestra el modelo completo usando *QF_BV* con codificación binaria para el *TSP*:

- **Parámetros:**

n : Número de nodos, $n \in \mathbb{Z}^+$

n^* : Número de nodos (en binario), $n^* \in \mathbb{Z}_2^{\lceil \log_2(n) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

$D_{i,j}$: Distancia entre el nodo i y el nodo j , $D_{i,j} \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$

$D_{i,j}^*$: Distancia entre el nodo i y el nodo j (en binario), $D_{i,j}^* \in \mathbb{Z}_2^{\lceil \log_2(\max(D_{i,j})) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$

- **Variables:**

$$X_{i,j} = \begin{cases} 1_2 & \text{si el nodo } j \text{ es visitado desde el nodo } i, \\ & X_{i,j} \in \mathbb{Z}_2^1 : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\} \\ 0_2 & \text{en caso contrario} \end{cases}$$

u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}_2^{\lceil \log_2(n) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}$

■ **Función objetivo:**

Minimizar distancia total recorrida $d = \sum_{i=1}^n \sum_{j=1}^n X_{i,j} \times D_{i,j}^*$

■ **Restricciones del problema:**

1. El dominio de $X_{i,j}$ está restringido entre 1₂ y 0₂ (Notar que esta restricción se puede omitir en esta oportunidad dado que el dominio de $X_{i,j}$ solo le permite tomar los valores 0 y 1 al tener un único bit disponible):

$$X_{i,j} \geq 0_2,$$

$$X_{i,j} \leq 1_2,$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

2. El dominio de u_i está restringido entre 1₂ y n^* :

$$u_i \geq 1_2,$$

$$u_i \leq n^*,$$

$$\forall i \in \{1, \dots, n\}$$

3. Cada nodo i debe ser un punto de partida una sola vez:

$$\sum_{j=1}^n X_{i,j} = 1_2, \forall i \in \{1, \dots, n\}$$

4. Cada nodo j debe ser punto de llegada una sola vez:

$$\sum_{i=1}^n X_{i,j} = 1_2, \forall j \in \{1, \dots, n\}$$

5. Eliminación de *sub-tours*:

$$u_i + n^* \times X_{i,j} \leq n^* - 1_2 + u_j, \forall i, j \in \mathbb{Z}^+ : 2 \leq i \leq n, 1 \leq j \leq n$$

6. Eliminación de *tours* con ciudades repetidas:

$$u_i \neq u_j, \forall i, j \in \{1, \dots, n\} : i \neq j$$

7. Un nodo no se puede visitar a sí mismo (*self-loop*):

$$X_{i,i} = 0_2, \forall i \in \{1, \dots, n\}$$

3.4.3. Transformación de LIA a QF_ALIA

La lógica de arreglos con extensionalidad permite modelar y resolver problemas usando estructuras con infinitos estados haciendo uso de algoritmos deductivos en *SMT* para controlar todos los estados activos existentes. La lógica original, *QF_AX*, define los arreglos como una colección de 3 elementos: un índice, un elemento y la estructura que los contiene (el mismo arreglo). Que los arreglos sean extensibles significa que los índices y los elementos de éste pueden pertenecer a cualquiera de los dominios definidos por las teorías *SMT*. Por otra parte, *QF_AX* por si solo no permite hacer más que realizar operaciones entre arreglos cuyos índices sean arreglos que apunten hacia otros arreglos, por lo que extender teorías como *LIA* o *LRA* para modelar problemas de programación lineal como los abordados en este trabajo resulta más natural y menos complejo que trabajar con teoría de arreglos pura.

Para este caso se optó por extender *LIA* para modelar y resolver los problemas planteados sin introducir mayores cambios a la lógica y estructura del modelo. Para ello, definimos un arreglo de la siguiente manera:

$\mathcal{A}_I^E = (\Sigma, \mathcal{C})$, siendo A la teoría de arreglos con índices I y elementos E . Σ y $\mathcal{C}$ representan aspectos formales que vinculan la teoría de índices con la teoría de elementos [68].

Con el fin de simplificar el extenso detalle técnico detrás de esta teoría, diremos que tanto los índices del arreglo como los elementos pertenecen al conjunto de los enteros, y que estos están ordenados dentro del arreglo, lo que permite trabajar asumiendo que un índice I_i siempre apuntará hacia el elemento E_i del arreglo $\mathcal{A}$, $\forall i \in \mathbb{Z}$.

Para hacer la transformación del modelo original es necesario convertir todas las variables en arreglos con índices y elementos en el dominio de los números enteros (*LIA*). Por ejemplo, consideremos la variable original del problema *TSP* modelado en la Sección 3.2.4:

- u_i : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u_i \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}$

El formato que usaremos para representar esta variable en *QF_ALIA* es la siguiente:

- u : variable auxiliar para definir el orden del *tour* y evitar *sub-tours*, $u \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

En el caso de las variables sin sub-índices, estas se transforman de la misma manera y al momento de modelar las restricciones se puede trabajar como un arreglo de tamaño 1.

En el caso de las variables con mas de un sub-índice, es posible representarlas usando arreglos multidimensionales, de manera que el primer índice apunte hacia un arreglo,

y que el índice de ese arreglo apunte hacia un valor numérico o hacia otro arreglo, dependiendo de lo especificado en el modelo original.

La teoría de arreglos también introduce cambios en la notación de las restricciones. Por ejemplo, una restricción del tipo:

$$x_{i,j} \geq 5, \forall i \in \{1, \dots, 5\}, \forall j \in \{1, \dots, 5\}$$

se transforma en la siguiente restricción [69]:

$$\text{select}(\text{select}(x, i), j) \geq 5, \forall i \in \{1, \dots, 5\}, \forall j \in \{1, \dots, 5\}$$

Por el bien de mantener consistencia en el formato de los modelos construidos, se decidió cambiar la notación por la utilizada en los lenguajes de programación. Se podría seguir usando notación de sub-índices, pero eso haría que sea complicado distinguir los modelos QF_ALIA de LIA dada la sintaxis similar.

Notación elegida para modelar restricciones con QF_ALIA:

$$x[i][j] \geq 5, \forall i \in \{1, \dots, 5\}, \forall j \in \{1, \dots, 5\}$$

A continuación se muestra el modelo completo usando QF_ALIA para el *TSP*:

- **Parámetros:**

n : Número de nodos, $n \in \mathbb{Z}^+$

$D_{i,j}$: Distancia entre el nodo i y el nodo j , $D_{i,j} \in \mathbb{Z}_0^+, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$

- **Variables:**

X : Arreglo de enteros bidimensional, almacena el valor 1 en E^* si se viaja de la ciudad I a la ciudad I^* , almacena el valor 0 en caso contrario, $X \in \mathcal{A}_I^E, \forall I \in \mathbb{Z}, E \in \mathcal{A}_{I^*}^{E^*}, \forall I^*, E^* \in \mathbb{Z}$

u : Arreglo auxiliar de enteros unidimensional, define el orden del *tour* y evita *sub-tours*, $u \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

- **Función objetivo:**

Minimizar distancia total recorrida $d = \sum_{i=1}^n \sum_{j=1}^n X[i][j] \times D_{i,j}$

- **Restricciones del problema:**

1. El dominio de $X[i][j]$ está restringido entre 1 y 0:

$$X[i][j] \geq 0,$$

$$X[i][j] \leq 1,$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

2. El dominio de $u[i]$ está restringido entre 1 y n :

$$u[i] \geq 1,$$

$$u[i] \leq n,$$

$$\forall i \in \{1, \dots, n\}$$

3. Cada nodo i debe ser un punto de partida una sola vez:

$$\sum_{j=1}^n X[i][j] = 1, \forall i \in \{1, \dots, n\}$$

4. Cada nodo j debe ser punto de llegada una sola vez:

$$\sum_{i=1}^n X[i][j] = 1, \forall j \in \{1, \dots, n\}$$

5. Eliminación de *sub-tours*:

$$u[i] - u[j] + n \times X[i][j] \leq n - 1, \forall i, j \in \mathbb{Z}^+ : 2 \leq i \leq n, 1 \leq j \leq n$$

6. Eliminación de *tours* con ciudades repetidas:

$$u[i] \neq u[j], \forall i, j \in \{1, \dots, n\} : i \neq j$$

7. Un nodo no se puede visitar a sí mismo (*self-loop*):

$$X[i][j] = 0, \forall i \in \{1, \dots, n\}$$

A partir de los experimentos realizados en el Capítulo 3 se construyeron 3 tablas por cada problema junto a sus respectivos gráficos. Una de las tablas tiene registro de los tiempos que duró la ejecución de cada *solver* para cada lógica por instancia, otra tabla contiene el registro de consumo de memoria (*RAM*) máximo que tuvo cada *solver* durante la ejecución, y la última tabla registra las soluciones encontradas por cada *solver*.

4.1. Optimización de modelos y aspectos técnicos

Para resolver y buscar las soluciones óptimas con los modelos construidos en la Sección 3.2 usando las instancias descritas en la Sección 3.1.1 se utiliza *Z3*, el *solver* de *Microsoft* [15] y su *API* para *Python* [69] para procesar cada instancia y generar automáticamente el código correspondiente a los modelos construidos siguiendo el formato de la *SMT-LIB 2.6* [18].

Al momento de generar estos archivos se ingresan las siguientes configuraciones:

- **Info:**
 - **status unknown:** Es la forma de decirle explícitamente al *solver* que queremos iniciar la búsqueda desde un estado inicial desconocido o no *sat*.
- **Logic:**
 - **LIA, LRA, QF_BV, ó QF_ALIA,** dependiendo de la lógica *SMT* utilizada en el modelamiento del problema.

Luego de tener los archivos con código en formato *SMT2* se usa la aplicación por consola de *Z3* para ejecutarlos y dejar que el *solver* busque la solución óptima usando las herramientas internas especializada en resolver problemas de *Optimization Modulo Theories* (*OMT*).

Al momento de ejecutar el comando de línea de *Z3* se incluyen los siguientes argumentos para rescatar información del proceso:

- **-smt2**: Indica que el archivo ingresado está en el formato estándar de *SMT2*
- **-st**: Es un *flag* para pedirle al *solver* que incluya un pequeño resumen técnico al finalizar la ejecución. En este resumen se incluye información relevante como el tiempo empleado y la cantidad de memoria utilizada durante la ejecución
- **-model**: Es un *flag* para pedirle al *solver* que incluya el modelo con los valores encontrados una vez que finalice la ejecución, dado que por defecto, *Z3* solo retorna **sat** si es que encuentra solución, y **unsat** en caso contrario

Adicionalmente, se fijó manualmente un tiempo máximo de ejecución de 24 horas o 86400 segundos. Una vez transcurrido ese tiempo (o una cantidad lo suficientemente cercana) se cancela cualquier *solver* que siga en ejecución y se registran las soluciones parciales encontradas, si es que existen.

El computador que se utilizó para obtener los resultados presentados a continuación cuenta con las siguientes características:

- Sistema operativo: Windows 11
- Versión de *Z3*: 4.12.2
- Procesador: i7-8700k
- Memoria disponible: 39.9 GB - 3200 MHz - DDR4

4.2. Calidad de las soluciones encontradas

Una manera para medir la calidad de las soluciones es viendo si un modelo logra encontrar una solución factible, y viendo que tan cerca del óptimo está dicha solución.

Viendo las soluciones obtenidas al resolver los problemas planteados en el Capítulo 3 se puede ver que para los problemas *NSP* y *N-queens* (Tablas 4.1 y 4.2, respectivamente) los modelos basados en **LIA** y **QF_BV** son los únicos que logran encontrar soluciones para todas las instancias, y en el caso de *NSP*, la diferencia que tienen las soluciones encontradas para las últimas instancias con el óptimo son menores al 1 %.

Instancias	LIA	LRA	QF_BV	QF_ALIA	Óptimo
25_1_g00	514	514	514	514	514
50_1_g00	960	960	960	960	960
75_1_g00	1432	1432	1432	1432	1432
100_1_g00	2211	X	2211	2211	2211
30_1_g00	2638	X	2639	2627	2639
60_1_g00	5130	X	5120	X	5130

Tabla 4.1: Tabla generada con las soluciones obtenidas por los modelos para el *NSP*. La primera columna contiene los nombres de las instancias resultas. El resto de las columnas contienen las soluciones encontradas por cada modelo, mientras que la última columna muestra el valor óptimo conocido. Las mejores soluciones (mayor valor) se distinguen por tener letra más oscura. Los casos en donde no se encuentra solución se marcan con una *X*. Fuente: Elaboración propia.

Instancias (n =)	LIA	LRA	QF_BV	QF_ALIA
4	1	1	1	1
8	1	1	1	1
12	1	1	1	1
16	1	1	1	1
20	1	1	1	1
50	1	1	1	1
100	1	X	1	1
120	1	X	1	1
140	1	X	1	X
160	1	X	1	X
180	1	X	1	X
200	1	X	1	X

Tabla 4.2: Tabla generada con las soluciones obtenidas por los modelos para el problema *N-queens*. La primera columna contiene los nombres de las instancias resultas. El resto de las columnas contienen un 1 si el modelo fue capaz de resolver el problema y una *X* en caso contrario. Fuente: Elaboración propia.

Otro problema donde se puede ver a los modelos basados en **QF_BV** teniendo problemas es *UKP*. Como se puede ver en la Tabla 4.3, **QF_BV** solo logra encontrar el óptimo en la primera instancia, y en comparación a los modelos basados en otras lógicas, la calidad de las soluciones encontradas por **QF_BV** son las únicas que decaen a medida que aumenta la complejidad de las instancias.

En el caso de *TSP*, al observar la Tabla 4.4 se puede ver que **QF_BV** logró encontrar las mejores soluciones hasta la cuarta instancia y luego falló en la última, mientras que, pese a que **LRA** y **QF_ALIA** generaron soluciones de peor calidad que **QF_BV**, fueron los únicos modelos que lograron encontrar al menos una solución factible para la última instancia,

Instancias	LIA	LRA	QF_BV	QF_ALIA	Óptimo
n10c7370	25727	25727	25727	25727	25727
n20c17050	42498	42498	33056	42498	42498
n30c27795	58350	58350	49547	58350	58350
n40c39588	94603	94603	57315	94603	94603
n50c41934	146750	146750	111595	146750	146750
n60c57601	192888	192888	134535	192888	192888
n70c68135	259939	259981	190996	259981	259981
n80c67603	263670	263670	X	263670	263670

Tabla 4.3: Tabla generada con las soluciones obtenidas por los modelos planteados para el *UKP*. La primera columna contiene los nombres de las instancias resultas. El resto de las columnas contienen las soluciones encontradas por cada modelo, mientras que la última columna muestra el valor óptimo conocido. Las mejores soluciones (mayor valor) se distinguen por tener letra más oscura. Los casos en donde no se encuentra solución se marcan con una *X*. Fuente: Elaboración propia.

independiente de lo lejos que se encuentren del óptimo. En esta tabla también se puede observar que LIA solo pudo encontrar la solución óptima de la primera instancia y las demás soluciones encontradas están más lejos del óptimo que las de los otros modelos.

Instancias	LIA	LRA	QF_BV	QF_ALIA	Óptimo
five	19	19	19	19	19
p01	347	291	291	291	291
gr17	3435	2085	2085	2388	2085
fri26	3080	2111	994	2421	937
att48	X	159096	X	143013	33551

Tabla 4.4: Tabla generada con las soluciones obtenidas por los modelos planteados para el *TSP*. La primera columna contiene los nombres de las instancias resultas. El resto de las columnas contienen las soluciones encontradas por cada modelo, mientras que la última columna muestra el valor óptimo conocido. Las mejores soluciones (menor valor) se distinguen por tener letra más oscura. Los casos en donde no se encuentra solución se marcan con una *X*. Fuente: Elaboración propia.

También hay problemas en los que los modelos basados en LIA y en QF_BV no logran encontrar solución para todas las instancias, como se puede ver en la Tabla 4.5 de BACP, donde los modelos basados en LRA son los únicos que logran encontrar el óptimo en todas las instancias, mientras que QF_BV presenta diferencias superiores al 40 % con respecto al óptimo de la primera y la tercera instancia.

Instancia	LIA	LRA	QF_BV	QF_ALIA	Óptimo
bacp8	17	17	24	17	17
bacp10	14	14	15	14	14
bacp12	X	17	28	X	17

Tabla 4.5: Tabla generada con las soluciones obtenidas por los modelos planteados para el *BACP*. La primera columna contiene los nombres de las instancias resultas. El resto de las columnas contienen las soluciones encontradas por cada modelo, mientras que la última columna muestra el valor óptimo conocido. Las mejores soluciones (menor valor) se distinguen por tener letra más oscura. Los casos en donde no se encuentra solución se marcan con una *X*. Fuente: Elaboración propia.

4.3. Tiempo empleado en encontrar solución

Salta a la vista que en problemas como *N-queens*, se puede observar que, a partir de la Figura 4.1, los modelos basados en LIA y QF_BV logran resolver todas las instancias abordadas considerablemente rápido, en comparación con los modelos basados en LRA y QF_ALIA, los cuales comenzaron a tener problemas con instancias donde $n \geq 100$, llegando a cancelar la ejecución en varias de esta por llegar al tiempo límite establecido en la Sección 4.1, lo que resulta consistente con lo observado en la Tabla 4.2 que contiene las soluciones encontradas por los modelos para este problema.

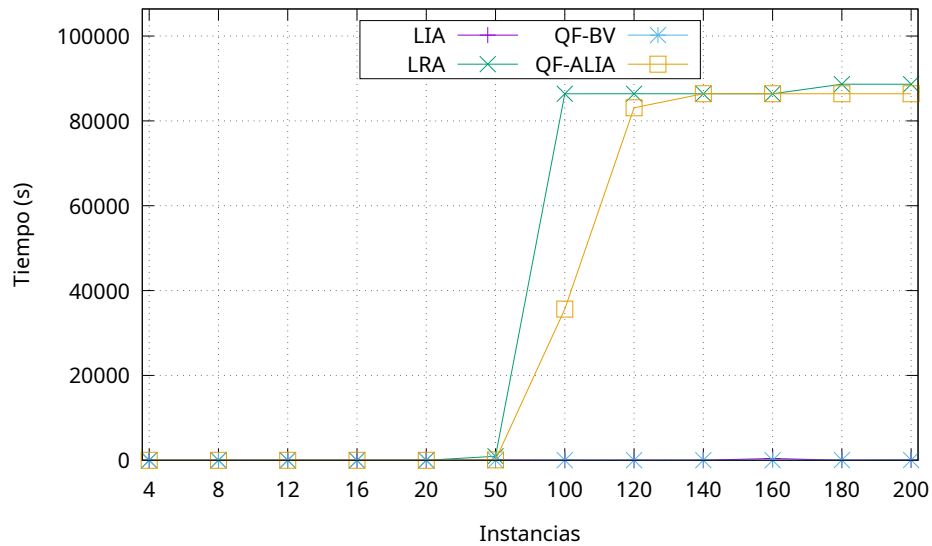


Figura 4.1: Gráfico con el tiempo total (en segundos) que duró la búsqueda de soluciones para cada uno de los modelos planteados para el problema *N-queens*. Los mejores tiempos se distinguen por ser más cercanos al cero en el eje Y. Fuente: Elaboración propia, a partir de la Tabla 5.1 registrada en Anexos.

En la Figura 4.2 se puede ver un efecto similar, en donde los modelos basados en LIA y QF_BV logran encontrar soluciones con un tiempo de ejecución menor al de los

otros modelos en las instancias más compleja (instancias representadas en la segunda mitad del eje X), mientras que para la última instancia evaluada, todos los modelos con excepción del basado en LIA presentan problemas y superan el tiempo máximo de ejecución definido en la Sección 4.1.

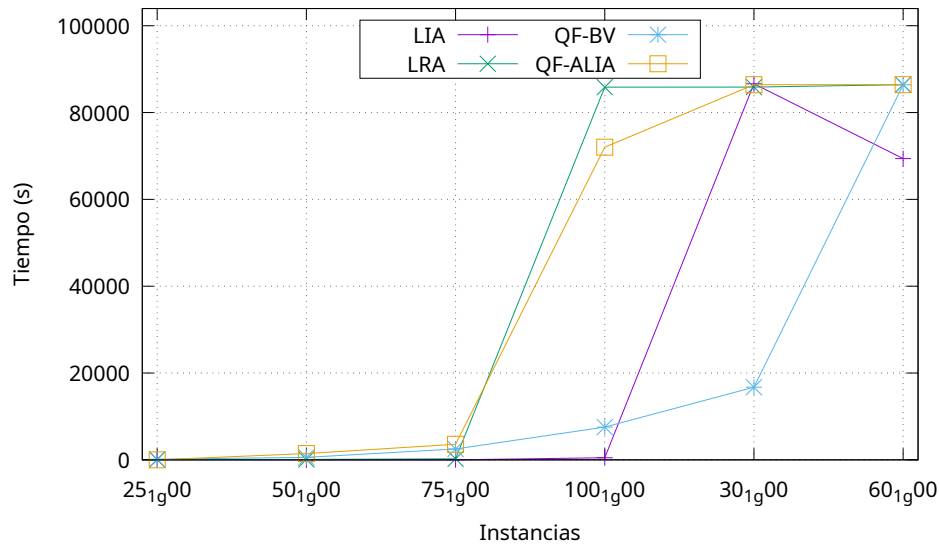


Figura 4.2: Gráfico con el tiempo total (en segundos) que duró la búsqueda de soluciones para cada uno de los modelos planteados para el problema *NSP*. Los mejores tiempos se distinguen por ser más cercanos al cero en el eje Y . Fuente: Elaboración propia, a partir de la Tabla 5.2 registrada en Anexos.

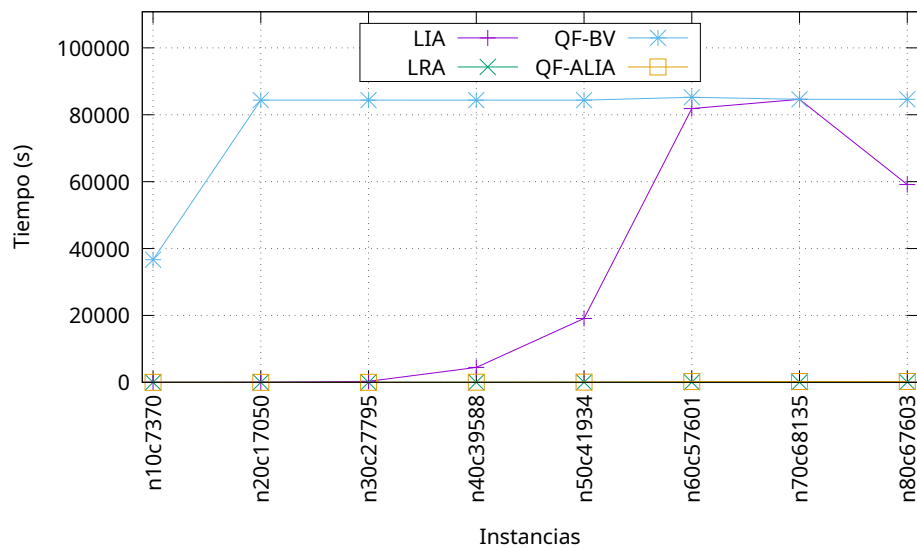


Figura 4.3: Gráfico con el tiempo total (en segundos) que duró la búsqueda de soluciones para cada uno de los modelos planteados para el problema *UKP*. Los mejores tiempos se distinguen por ser más cercanos al cero en el eje Y . Fuente: Elaboración propia, a partir de la Tabla 5.3 registrada en Anexos.

En cuanto a *UKP*, la Tabla 4.3 con las soluciones encontradas para este problema, vista en la Sección 4.2, da a entender que todos los modelos tuvieron un comportamiento similar, salvo por *QF_BV*. Al observar la Figura 4.3 con los tiempos de ejecución de los modelos es posible ver que, como se esperaba a partir de las soluciones no óptimas encontradas, los modelos basados en *QF_BV* superaron el tiempo máximo permitido constantemente a partir de la segunda instancia, mientras que los modelos *LIA*, *QF_ALIA* y *LRA* aumentan el tiempo requerido de cómputo a medida que aumenta la complejidad de las instancias, con la diferencia de que los tiempos requeridos por los modelos basados en *QF_ALIA* y *LRA* parecieran ser entre 1 y 2 ordenes de magnitud menor a los tiempos requeridos por *LIA*.

Por otra parte, en problemas como *TSP*, los modelos basados en *LIA* solo logran completar la primera instancia antes de ser cancelados, como se puede ver en la Figura 4.4, lo que justifica la baja calidad de las soluciones encontradas por *LIA* en la Tabla 4.4 con respecto a los otros modelos.

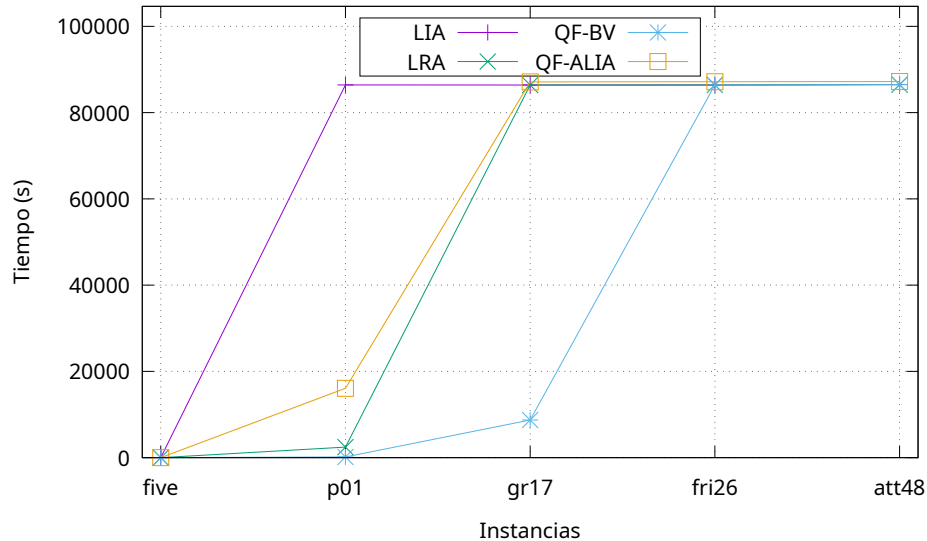


Figura 4.4: Gráfico con el tiempo total (en segundos) que duró la búsqueda de soluciones para cada uno de los modelos planteados para el problema *TSP*. Los mejores tiempos se distinguen por ser más cercanos al cero en el eje Y. Fuente: Elaboración propia, a partir de la Tabla 5.4 registrada en Anexos.

Hasta ahora, lo observado a partir de los resultados de los problemas anteriores parece indicar que a medida que aumenta la complejidad de las instancias, aumenta el tiempo requerido para resolverlas y, en algunos casos, empeora la calidad de las soluciones encontradas.

Sin embargo, al observar la Figura 4.5 con los tiempos de ejecución para *BACP*, se puede ver que dicho patrón solo está presente en *LIA*, *QF_ALIA* y *QF_BV*, mientras que *LRA* superó el tiempo permitido en las dos primeras instancias, las de menor complejidad, para luego resolver la tercera instancia, de mayor complejidad, en solo 10 segundos.

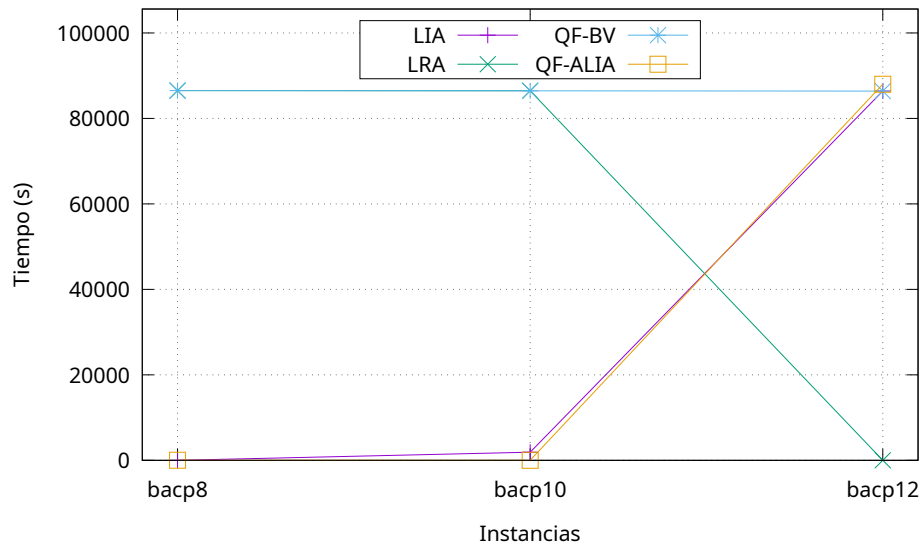


Figura 4.5: Gráfico con el tiempo total (en segundos) que duró la búsqueda de soluciones para cada uno de los modelos planteados para el problema *BACP*. Los mejores tiempos se distinguen por ser más cercanos al cero en el eje Y. Fuente: Elaboración propia, a partir de la Tabla 5.5 registrada en Anexos.

4.4. Uso de memoria

A nivel general se puede observar que, a partir de las Tablas 4.6, 4.10, 4.8, 4.7 y 4.9 con los registros de la máxima cantidad de memoria utilizada, los modelos basados en LIA son los que presentan consistentemente un menor consumo de memoria durante la búsqueda de la solución óptima a lo largo de todos los problemas, salvo por UKP.

Instancia	LIA	LRA	QF_BV	QF_ALIA
bacp8	52.44	752.04	2744.82	112.16
bacp10	71.88	642.82	1858.8	97.46
bacp12	183.6	1817.3	4757.64	196.76

Tabla 4.6: Tabla con los registros de la máxima memoria en *Mb* (*RAM*) utilizada por el *solver* al momento de buscar la solución óptima para el problema *BACP*. Los mejores niveles de consumo (menor valor por instancia) se distinguen por tener letra más oscura. Fuente: Elaboración propia.

En particular, llegó a utilizar cerca de 29 *GB* en una instancia de *TSP* y en una instancia de *NSP*. QF_ALIA también muestra un comportamiento similar, llegando a los 33 *GB* en una instancia de *TSP*. En tercer lugar se encuentra QF_BV cuyos peores casos registran un consumo de casi 21 y 15 *GB* con una instancia de *NSP* y una de *TSP*, respectivamente. En el caso de los modelos basados en LIA, estos presentan consistentemente el menor consumo de memoria, llegando a alcanzar un máximo de

consumo cercano a los 7 GB en una instancia de *NSP*.

Al igual que la relación entre calidad de solución - tiempo de ejecución, la cantidad de memoria consumida también aumenta a medida que aumenta el tiempo de ejecución.

Instancias	LIA	LRA	QF_BV	QF_ALIA
five	35.9	36.12	38.6	45.06
p01	81.5	135.64	426.86	102.26
gr17	135.2	245.48	603.96	114.34
fri26	313.68	1829.3	6529.7	316.74
att48	1277.8	28899.56	15988.56	2042.46

Tabla 4.7: Tabla con el consumo máximo de memoria en *Mb* (*RAM*) utilizada por el *solver* al momento de buscar la solución óptima para el problema *TSP*. Los mejores niveles de consumo (menor valor por instancia) se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Instancias	LIA	LRA	QF_BV	QF_ALIA
25_1_g00	117.68	479.54	1200.44	174.16
50_1_g00	255.94	1825.38	2447.62	1944.36
75_1_g00	685.26	3643.7	4725.76	2644.08
100_1_g00	1327.06	7240.4	5134.46	8972.16
30_1_g00	2048.92	7325.98	9774.6	7134.48
60_1_g00	6873.9	29348.76	20908.82	33906.24

Tabla 4.8: Tabla con el consumo máximo de memoria en *Mb* (*RAM*) utilizada por el *solver* al momento de buscar la solución óptima para el problema *NSP*. Los mejores niveles de consumo (menor valor por instancia) se distinguen por tener letra más oscura. Fuente: Elaboración propia.

En el caso particular de la Tabla 4.9 se puede ver como los modelos basados en *QF_ALIA* logran consumir menos memoria que los basados en *LIA*.

A simple vista puede parecer contradictorio si se compara con los registros de consumo de memoria en las otras tablas, pero en este caso hay que tener en consideración que, a partir de la Figura 4.3 con los tiempos de ejecución de *UKP*, la diferencia en los tiempos de ejecución alcanzados resulta clave en el uso de memoria final, siendo el tiempo requerido por *LIA* entre 1 y 2 ordenes de magnitud mayor al de *QF_ALIA* para este problema en particular, justificando el bajo uso de memoria registrado en *QF_ALIA* en comparación con *LIA*.

Instancias	LIA	LRA	QF_BV	QF_ALIA
n10c7370	44.28	54.34	646.6	45.38
n20c17050	78.2	52.54	1950.18	47.48
n30c27795	135.52	68.42	2898.48	49.04
n40c39588	252.46	88.62	2810.7	55.44
n50c41934	953.3	150.8	3496.88	62.28
n60c57601	1871.04	204.88	5327.2	82.88
n70c68135	953.1	310.5	5126.98	68.34
n80c67603	1879.8	329.72	7282.58	80.48

Tabla 4.9: Tabla con el consumo máximo de memoria en *Mb* (*RAM*) utilizada por el *solver* al momento de buscar la solución óptima para el problema *UKP*. Los mejores niveles de consumo (menor valor por instancia) se distinguen por tener letra más oscura. Fuente: Elaboración propia.

En el caso de la Tabla 4.10 se observa un fenómeno similar al descrito en *UKP*, solo que esta vez los modelos basados en LIA son los que tienen un menor consumo de memoria en comparación a QF_ALIA, y como se mencionó anteriormente, una explicación para este caso en particular puede ser nuevamente el tiempo de ejecución, dado que los modelos basados en LIA requirieron de una cantidad de tiempo entre 1 y 2 ordenes de magnitud menor a la de los modelos basados en QF_ALIA a partir de las instancias donde $n \geq 100$ en adelante, como se observó en la Figura 4.1.

Instancias (n =)	LIA	LRA	QF_BV	QF_ALIA
4	17.5	17.59	25.1	21.8
8	17.74	18.12	25.27	20.64
12	17.81	19.09	25.81	22.69
16	18.21	21.36	26.24	23.78
20	18.64	22.79	27.63	25.3
50	42.73	150.43	52.2	103.37
100	117.52	1149.95	223.15	1199.54
120	222.04	1586.94	287.14	1931.11
140	225.89	1903.66	460.49	2552.07
160	432.35	2111.01	558.76	3324.9
180	436.83	2795.55	688.15	4652.8
200	447.57	2669.91	1080.22	5473.08

Tabla 4.10: Tabla con los registros de la máxima memoria en *Mb* (*RAM*) utilizada por el *solver* al momento de buscar la solución óptima para el problema *N-queens*. Los mejores niveles de consumo (menor valor por instancia) se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Conclusiones

Por último, en este capítulo se provee un resumen del trabajo realizado a lo largo de la tesis junto a las conclusiones que pueden extraerse de ésta, señalando nuevas ideas que podrían considerarse en un estudio futuro.

En en la Sección 2.1 de este trabajo se abordaron problemas de optimización lineal conocidos y relevantes en el mundo real fuera de lo académico. Para formalizar estos problemas, en la Sección 2.3 se consideraron los estándares definidos para modelarlos como problemas compatibles con *SMT*, y en la Sección 2.4 se abordó la extensión de *SMT* que permite resolver problemas de optimización, *OMT*.

En total se modelaron 5 problemas de satisfacción de restricciones (y optimización) conocidos en la Sección 3.2: *N-queens*, *Balanced Academic Curriculum Problem*, *Nurse Scheduling Problem*, *Travelling Salesman Problem* y *Unbounded Knapsack Problem*, usando aritmética lineal de enteros o *LIA* como lógica *SMT*. Luego, se definieron reglas para traducir cada uno de estos modelos a modelos basados en 3 lógicas *SMT* distintas vistas en la Sección 3.4: *LRA*, *QF_BV* y *QF_AX* extendiendo la teoría de enteros (*QF_ALIA*).

Para cada problema se definió un conjunto con instancias de distinta complejidad en la Sección 3.1.1 para poder comparar el impacto que produce la complejidad sobre los modelos formalizados en distintas lógicas *SMT*.

Y por último, se usó *Z3* para resolver los modelos construidos a partir de distintas lógicas y buscar la solución óptima para cada instancia, dejando registro de la metodología y de los resultados observados en la Sección 4.

En particular, se evaluó las soluciones encontradas por cada modelo usando 3 criterios: el tiempo total de ejecución del *solver*, la cantidad máxima de memoria en uso registrada durante la ejecución y la calidad de las soluciones encontradas por cada modelo respecto a un óptimo conocido como referencia.

De los resultados registrados, la calidad de las soluciones y el tiempo de ejecución son los

que permiten visualizar mejor el comportamiento de los modelos construidos a partir de distintas lógicas, dado que en los casos donde dos o más modelos encuentren el óptimo, el factor diferenciable es el tiempo de ejecución, mientras que en los casos donde dos o más modelos superen el tiempo de ejecución máximo permitido (24 horas), el factor diferenciable será la calidad de las soluciones encontradas.

Para los problemas de optimización abordados en este documento se puede afirmar que efectivamente hay lógicas *SMT* que al ser usadas para modelar un determinado problema permiten obtener mejores soluciones que los modelos basados en otras lógicas, y también se puede ver el hecho de que las ventajas en desempeño de un modelo basado en una determinada lógica no necesariamente se replicará al momento de cambiar a otro problema.

Por ejemplo, en base a lo observado en los resultados para los problemas *NSP* y *N-queens* sería normal pensar que los modelos basados en *LIA* o en *QF_BV* son superiores a *LRA* o a *QF_ALIA* dado que los primeros estuvieron bastante cerca del óptimo (con una diferencia menor al 1% en los peores casos), mientras que los últimos dos empezaron a tener problemas con la segunda mitad de las instancias, especialmente *LRA* que fue cancelado sin encontrar ninguna solución para esos casos. Sin embargo, la situación cambia en problemas como *UKP*, donde si bien todos los modelos menos el basado en *QF_BV* lograron soluciones óptimas, los tiempos de ejecución detrás de cada búsqueda dan a entender que los modelos basados en *LIA* se encuentran muy cerca del tiempo máximo de ejecución, especialmente en las últimas instancias, y en el caso de *QF_BV*, éste comenzó a superar el tiempo de ejecución permitido a partir de la segunda instancia, mientras que *LRA* y *QF_ALIA* requieren entre 100 y 300 veces menos tiempo para encontrar la solución óptima, lo que les da una ventaja significativa en caso de que tuvieran que seguir resolviendo instancias más complejas de *UKP*.

Acerca de posibles patrones que den algún indicio previo sobre el desempeño que tendrá un modelo para un determinado problema, al analizar los tipos de restricciones que definen cada problema en la Sección 3.3 se puede ver que *UKP* solo posee variables numéricas y estas son acotadas inferior y superiormente, y que su función objetivo busca maximizar la suma de éstas. El problema que más se le parece en cuanto a tipo de restricciones implementadas es *N-queens* dado que también posee una única variable numérica acotada, sin embargo, los resultados observados para ambos problemas fueron totalmente opuestos en cuanto a las lógicas que permiten llegar a mejores soluciones y tiempos.

Los otros dos problemas que presentan similitud en sus restricciones y función objetivo son *TSP* y *BACP*, y los resultados obtenidos indican que en *BACP*, las mejores lógicas fueron *LRA* seguida por *LIA* y *QF_ALIA*, siendo *QF_BV* la que más se alejó del óptimo, mientras que en *TSP*, las mejores lógicas en general fueron *LRA* y *QF_BV*.

Estas observaciones muestran que la existencia de un patrón no es distinguible enteramente cuando se modela usando una sola lógica, lo que motiva aún más la idea de que los modelos se construyan a partir de múltiples lógicas dependiendo de los tipos y de la cantidad de variables que tenga un determinado problema, de esa manera la lista de

restricciones asignada a cada lógica será menor y será posible realizar un análisis más fino sobre el impacto que produce cada lógica

En esta oportunidad se pudo resolver todos los problemas modelados usando la máquina descrita en la Sección 4.1, sin embargo, es sabido que la cantidad de recursos varía de una máquina a otra y, dependiendo de los requerimientos, muchas veces se vuelve necesario poder ejecutar estos modelos en dispositivos con una cantidad limitada de recursos (por ejemplo, en sistemas *embedded*), por lo que los registros de memoria RAM utilizada sirven como guía general para elegir un modelo por sobre otro cuando las decisiones de diseño pesan más que la necesidad de encontrar una solución óptima. En ese sentido, se puede ver que las diferencias en el uso de memoria entre los modelos basados en distintas lógicas es relativamente consistente a lo largo de los problemas resueltos. En particular, los modelos basados en LIA suelen tener un menor consumo en comparación a LRA, QF_ALIA y QF_BV. Estos últimos, dependiendo del tiempo de ejecución pueden llegar a consumir una cantidad importante de recursos como se pudo ver en los resultados de *TSP* y *NSP*.

Sobre los objetivos mencionados al inicio de este documento en la Sección 1.1, se logró modelar y resolver satisfactoriamente no solo 3 problemas de satisfacción de restricciones usando 3 lógicas distintas, sino que 5 problemas usando 4 lógicas distintas sobre un relativamente amplio conjunto de instancias considerando el contexto y los recursos bajo los que se llevó a cabo el desarrollo de esta tesis. Ampliar el alcance planificado inicialmente permitió añadir más variedad a la comparación de resultados obtenidos.

También se logró el objetivo de evaluar y comparar el rendimiento de los problemas resueltos como se ha ido mencionando a lo largo de este capítulo.

Si bien se eligieron dos problemas en los que se podría llevar a cabo una combinación de lógicas en el modelamiento (*BACP* y *TSP* dado que tienen más de un tipo de variable en el modelamiento), el estudio empírico se centró únicamente en incorporar una sola de las lógicas elegidas por modelo. Considerar que combinar n lógicas en un problema con k tipos de variables implicaría resolver ese problema k^n veces, razón por lo que queda propuesto como trabajo futuro ver el desempeño que estos modelos mixtos tengan en comparación con los modelos puros al momento de resolver el mismo problema.

En cuanto a las contribuciones de este trabajo, el hecho de presentar evidencia empírica sobre la diferencia de desempeño en modelos basados en distintas lógicas abre las puertas a la posibilidad de conseguir mejores resultados en casi cualquier modelo ya construido y estudiado en cualquier área de la optimización, el “único” cambio que necesitarían los modelos ya existentes sería la traducción hacia alguna otra lógica que le genere mayores beneficios, entendiendo que la dificultad de esa traducción dependerá enteramente del modelamiento inicial y el dominio que abarca.

También se presentó una serie de reglas que permite tomar modelos ya existentes de programación lineal entera o LIA y traducirlos a 3 lógicas *SMT* distintas: LRA, QF_BV y QF_AX con extensión de números enteros. Estas reglas permiten cambiar el dominio de estos modelos sin pérdida de generalidad, es decir, sin alterar el conjunto de soluciones

encontradas y los posibles valores disponibles para cada variable.

Las limitaciones de esta propuesta se pueden resumir en la carencia de un patrón o conjunto de reglas que permita determinar a priori cuál lógica producirá un mejor desempeño. Es por ello que, en vez de ofrecer una solución universal que lo mejore todo, esta solución plantea que existe un hiper-parámetro con valores finitos (lógicas *SMT*) que se puede ajustar al momento de construir y evaluar modelos para *CSP/CSOP*.

5.1. Trabajos futuros

En la propuesta de solución para esta tesis se eligieron problemas cuyos parámetros y soluciones pertenecen al dominio de los números enteros. Esto facilita la traducción hacia las otras lógicas dado que la lógica de *bit-vectors* y de *array* poseen un dominio casi equivalente al de los números enteros, y en el caso de *LRA*, el dominio de los números reales contiene al de los números enteros, por lo que la traducción del contexto del problema se puede hacer de manera directa.

Una comparación que podría ser interesante en el futuro es la de incorporar nuevos problemas con dominios basados en distintas teorías. Esto podría generar algunos conflictos al momento de traducir hacia otras lógicas, especialmente si el dominio inicial es el de los números reales dada la posible pérdida de información al acotar el problema al dominio de los números enteros, pero podría generar nuevos resultados y conclusiones que no se hayan considerado en el estudio realizado en este documento. También sería de interés ver que tipo de comparaciones se pueden obtener al resolver problemas de optimización combinatoria o no lineal haciendo uso de distintas lógicas.

Otro aspecto interesante que no se llevó a cabo empíricamente en este trabajo es el de combinar más de una lógica en un mismo modelo. De los problemas que se modelaron en la Sección 3.2, *BACP* y *TSP* contienen más de una variable, por lo que existe la opción de modelar cada variable usando distintas lógicas para analizar como se comporta durante la búsqueda de soluciones. En los resultados obtenidos se vio que algunos modelos basados en determinadas lógicas logran resolver mejor un problema que otros, por lo que la idea de ajustar los modelos desde la capa de las variables en vez de ajustarlo a nivel del problema completo podría dar paso a un afinamiento más profundo que permita obtener aún mejores resultados en determinados problemas en comparación a modelamientos basados en una sola lógica.

También resultaría interesante ver como se comportan los modelos basados en las lógicas que no se vieron en este documento, como *Integer/Rational Difference Logic* o *Floating Point* o *Arrays* puros, entre otras.

Si analizamos el estudio realizado en este documento desde el punto de vista de la automatización de procesos salta a la vista que un trabajo a futuro puede ser el de automatizar la configuración de lógicas que forman parte de un determinado modelamiento, ya sea en base a un conjunto de reglas definidas a partir de resultados empíricos

o en base a técnicas de aprendizaje automático y/o inteligencia artificial. Siguiendo la línea de automatización, también sería interesante como trabajo futuro el desarrollo de un framework que permita traducir automáticamente el modelamiento de un problema para que integre otras lógicas *SMT*. Una idea para lograrlo podría ser basarse en el framework de *SMT* ya que este se encuentra constantemente transformando el modelo entre su configuración inicial y *SAT*, pasando de un dominio a otro en cada iteración.

Anexos

Transformaciones de LIA a LRA

N-queens

- **Parámetros:**

N : Número de reinas que se deben posicionar en un tablero de tamaño $N \times N$, tal que $N \in \mathbb{Z}^+$

- **Variables:**

X_c : Número de la fila en la columna c donde se posiciona una reina, $X_c \in \mathbb{R}, \forall c \in \{1, \dots, N\}$

- **Función objetivo:**

Que ninguna reina en el tablero pueda ser atacada, o en otras palabras, que se cumplan todas las restricciones del problema

- **Restricciones del problema:**

1. Acotamiento del dominio de X_c :

$$\bigvee_{j=1}^n X_c = j, \forall c \in \{1, \dots, N\}$$

2. El dominio de X_c está restringido entre 1 y n (cantidad de filas) (Notar que esta restricción se puede omitir dado que ya está considerada dentro del acotamiento anterior, sin embargo se decidió dejarla por completitud):

$$X_c \geq 1,$$

$$X_c \leq N,$$

$$\forall c \in \{1, \dots, N\})$$

3. Debe haber a lo mas una reina por fila:

$$X_{c_i} \neq X_{c_j}$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_i + 1, \dots, N\}$$

4. Debe haber a lo mas una reina por diagonal

$$X_{c_j} \neq X_{c_i} + (c_j - c_i)$$

$$X_{c_i} \neq X_{c_j} + (c_j - c_i)$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_i + 1, \dots, N\}$$

Balanced Academic Curriculum Problem (BACP)

■ **Parámetros:**

m : Número de cursos, $m \in \mathbb{Z}^+$

n : Número de periodos académicos, $n \in \mathbb{Z}^+$

α_i : Número de créditos del curso i , $\alpha_i \in \mathbb{Z}^+$, $\forall i \in \mathbb{Z}^+ : i \in \{1, \dots, m\}$

β : Carga académica **mínima** permitida por periodo, $\beta \in \mathbb{Z}^+$

γ : Carga académica **máxima** permitida por periodo, $\gamma \in \mathbb{Z}^+$

δ : Cantidad de cursos **mínima** por periodo, $\delta \in \mathbb{Z}^+$

ϵ : Cantidad de cursos **máxima** por periodo, $\epsilon \in \mathbb{Z}^+$

■ **Variables:**

$$X_{i,j} = \begin{cases} 1 & \text{si el curso } i \text{ es asignado al periodo } j, \\ & X_{i,j} \in \mathbb{R}, \forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\} \\ 0 & \text{en caso contrario} \end{cases}$$

c_j : carga académica del periodo j , $c_j \in \mathbb{Z}, \forall j \in \{1, \dots, n\}$

C : Mayor carga académica registrada a lo largo del currículum, $C \in \mathbb{Z}$

■ **Función objetivo:**

Minimizar la carga académica a lo largo de los periodos, lo que se puede representar como: minimizar la mayor carga académica registrada a lo largo del currículum:

Minimizar C

■ **Restricciones del problema:**

1. Acotamiento del dominio de $X_{i,j}$:

$$\bigvee_{k=0}^1 X_{i,j} = k, \forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\}$$

2. Se podría acotar c_j , pero sus valores se definen a partir de $X_{i,j}$ y α_i , por lo que no es necesario.
3. El dominio de $X_{i,j}$ está restringido entre 0 y 1 (se puede omitir dado que ya está considerado en la restricción 1):

$$X_{i,j} \leq 1,$$

$$X_{i,j} \geq 0,$$

$$\forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\}$$

4. La mayor carga académica C se define como:

$$C = \text{Max}\{c_1, \dots, c_n\}$$

Lo que se puede forzar usando la siguiente restricción lineal en conjunto con la función objetivo:

$$C \geq c_j, \forall j \in \{1, \dots, n\}$$

5. La carga académica para el periodo j se define como:

$$c_j = \sum_{i=1}^m \alpha_i \times X_{i,j}, \forall j \in \{1, \dots, n\}$$

6. Cada curso i debe ser asignados a un periodo j :

$$\sum_{j=1}^n X_{i,j} = 1, \forall i \in \{1, \dots, m\}$$

7. Curso b tiene como pre-requisito al curso a , por lo tanto el curso b no se puede asignar a un periodo anterior al asignado para el curso a :

$$X_{b,j} \leq \sum_{r=1}^{j-1} X_{a,r} = 1, \forall j \in \{2, \dots, n\}, \forall a \in \{1, \dots, m-1\}, \forall b \in \{a, \dots, m\}$$

8. La carga académica de cada periodo j debe ser mayor o igual a la carga mínima permitida:

$$c_j \geq \beta, \forall j \in \{1, \dots, n\}$$

9. La carga académica de cada periodo j debe ser menor o igual a la carga máxima permitida:

$$c_j \leq \gamma, \forall j \in \{1, \dots, n\}$$

10. El número de cursos del periodo j debe ser mayor o igual a la cantidad mínima permitida:

$$\sum_{i=1}^m X_{i,j} \geq \delta, \forall j \in \{1, \dots, n\}$$

11. El número de cursos del periodo j debe ser menor o igual que la cantidad máxima permitida:

$$\sum_{i=1}^m X_{i,j} \leq \epsilon, \forall j \in \{1, \dots, n\}$$

Nurse Scheduling Problem (NSP)

■ **Parámetros:**

N : Número de *nurses*, $N \in \mathbb{Z}^+$

D : Número de días que definen la jornada de trabajo, $D \in \mathbb{Z}^+$

S : Número de turnos por día (el último no turno se considera como turno de trabajo, es decir, si $S = 4$, el día contará con $S - 1 = 3$ turnos laborales), $S \in \mathbb{Z}^+$

$C_{d,s}$: Matriz de cobertura, indica la cantidad de *nurses* requerida para el día d y el turno s , $C_{d,s} \in \mathbb{Z}_0^+, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$P_{n,d,s}$: Matriz de preferencias, indica la preferencia que tiene un *nurse* n para trabajar el día d en el turno s , $P_{n,d,s} \in \mathbb{Z}_0^+, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

min_wd : Cantidad mínima de días en que a cada *nurse* se le debe asignar un turno laboral, $min_wd \in \mathbb{Z}_0^+$

max_wd : Cantidad máxima de días en que a cada *nurse* se le debe asignar un turno laboral, $max_wd \in \mathbb{Z}_0^+$

min_cd : Cantidad mínima de días de trabajo consecutivos, $min_cd \in \mathbb{Z}_0^+$

max_cd : Cantidad máxima de días de trabajo consecutivos, $max_cd \in \mathbb{Z}_0^+$

min_cs_s : Número entero que indica la cantidad mínima de días consecutivos que un *nurse* deberá trabajar en el turno s , en caso de que se le sea asignado, $min_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

max_cs_s : Número entero que indica la cantidad máxima de días consecutivos que un *nurse* podrá trabajar en el turno s , $max_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

min_s_s : Número entero que indica la cantidad mínima de veces que se debe asignar el turno s a cada *nurse*, $min_s_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

max_s_s : Número entero que indica la cantidad máxima de veces que se puede asignar el turno s a cada $nurse$, $max_s_s \in \mathbb{Z}_0^+$, $s \in \{1, \dots, S\}$

■ **Variables:**

$$X_{n,d,s} = \begin{cases} 1,0 & \text{si el } nurse \ n \text{ es asignado al día } d \text{ en el turno } s, \\ & X_{n,d,s} \in \mathbb{R}, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\} \\ 0,0 & \text{en caso contrario} \end{cases}$$

■ **Función objetivo:**

Maximizar $H = \sum_{n=1}^N \sum_{d=1}^D \sum_{s=1}^S X_{n,d,s} \times P_{n,d,s}$ (*Happiness*, determinado por los turnos asignados a cada nurse y su preferencia sobre estos), $\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

■ **Restricciones del problema:**

1. Acotamiento del dominio de $X_{n,d,s}$:

$$\bigvee_{k=0}^1 X_{n,d,s} = k, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$$

2. El dominio de $X_{n,d,s}$ está restringido entre 1 y 0 (se puede omitir dado que ya está considerado en la restricción 1):

$$X_{n,d,s} \geq 0,0,$$

$$X_{n,d,s} \leq 1,0,$$

$$\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$$

3. Cada $nurse$ n debe tener asignado un turno s cada día:

$$\sum_{s=1}^S X_{n,d,s} = 1,0, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}$$

4. Cada turno laboral s en cada día d deberá tener al menos la cantidad $C_{d,s}$ de $nurses$ indicada en la Matriz de cobertura:

$$\sum_{n=1}^N X_{n,d,s} \geq C_{d,s}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S-1\}$$

5. A cada nurse se le debe asignar al menos min_wd y a lo más max_wd días de trabajo (sin contar el turno libre):

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \geq min_wd, \forall n \in \{1, \dots, N\}$$

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \leq max_wd, \forall n \in \{1, \dots, N\}$$

6. Cada *nurse* debe tener asignado un determinado turno s al menos min_s_s y a lo más max_s_s veces:

$$\sum_{d=1}^D X_{n,d,s} \geq min_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

$$\sum_{d=1}^D X_{n,d,s} \leq max_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

7. Cada *nurse* debe trabajar al menos min_cd y a lo más max_cd días consecutivos

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} \sum_{s=1}^{S-1} X_{n,j-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i+k-1,s} \leq k - 2,$$

$$\forall n \in \{1, \dots, N\}, \forall k \in \{\min\{2, min_cd\}, \dots, min_cd+1\}, \forall i \in \{1, \dots, D+2-k\}$$

$$\sum_{d=j}^{\min\{D,j+max_cd+1\}} \sum_{s=1}^{S-1} X_{n,d,s} \leq max_cd,$$

$$\forall n \in \{1, \dots, N\}, \forall j \in \{1, \dots, D - max_cd\}$$

8. Cada *nurse* debe repetir el mismo turno de trabajo s al menos min_cs_s y a lo más max_cs_s veces, en caso de que el turno s le sea asignado

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} X_{n,j-1,s} - 2 \times X_{n,i-1,s} - 2 \times X_{n,i+k-1,s} \leq k - 2,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S - 1\},$$

$$\forall k \in \{\min\{2, min_cs_s\}, \dots, min_cs_s + 1\}, \forall i \in \{1, \dots, D + 2 - k\}$$

$$\sum_{d=j}^{\min\{D,j+max_cs_s+1\}} X_{n,d,s} \leq max_cs_s,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S - 1\}, \forall j \in \{1, \dots, D - max_cd\}$$

Unbounded Knapsack Problem (UKP)

■ Parámetros:

n : Número entero con la cantidad total de objetos, $n \in \mathbb{Z}^+$

C : Número entero con la capacidad máxima de la mochila, $C \in \mathbb{Z}^+$

W_i : Número entero indicando el peso del objeto i , $W_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

V_i : Número entero indicando el valor del objeto i , $V_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

- **Variables:**

X_i : cantidad de veces que se agrega el objeto i a la mochila, $X_i \in \mathbb{R}, \forall i \in \{1, \dots, n\}$

- **Función objetivo:**

Maximizar valor v en la mochila, $v = \sum_{i=1}^n V_i \times X_i$

- **Restricciones del problema:**

1. Acotamiento del dominio de X_i :

$$\bigvee_{j=1}^{\lceil \frac{C}{W_i} \rceil} X_c = j, \forall c \in \{1, \dots, n\}$$

2. Cada objeto puede ser ingresado cero o más veces:

$$X_i \geq 0, \forall i \in \{1, \dots, n\}$$

3. La suma de los pesos de los objetos ingresados no debe superar la capacidad máxima de la mochila:

$$\sum_{i=1}^n W_i \times X_i \leq C$$

Transformaciones de LIA a QF_BV

N-queens

- **Parámetros:**

N : Número de reinas que se deben posicionar en un tablero de tamaño $N \times N$, tal que $N \in \mathbb{Z}^+$

N^* : Número de reinas que se deben posicionar en un tablero de tamaño $N \times N$ (en binario), tal que $N \in \mathbb{Z}_2^{\lceil \log_2(N) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

- **Variables:**

X_c : Número de la fila en la columna c donde se posiciona una reina, $X_c \in \mathbb{Z}_2^{\lceil \log_2(N) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall c \in \{1, \dots, N\}$

- **Función objetivo:**

Que ninguna reina en el tablero pueda ser atacada, o en otras palabras, que se cumplan todas las restricciones del problema

- **Restricciones del problema:**

1. El dominio de X_c está restringido entre 1 y N (cantidad de filas):

$$X_c \geq 1_2,$$

$$X_c \leq N^*,$$

$$\forall c \in \{1, \dots, N\}$$

2. Debe haber a lo mas una reina por fila:

$$X_{c_i} \neq X_{c_j}$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_1 + 1, \dots, N\}$$

3. Debe haber a lo mas una reina por diagonal

$$X_{c_j} \neq X_{c_i} + (c_j - c_i)_2$$

$$X_{c_i} \neq X_{c_j} + (c_j - c_i)_2$$

$$\forall c_i \in \{1, \dots, N-1\}$$

$$\forall c_j \in \{c_1 + 1, \dots, N\}$$

Balanced Academic Curriculum Problem (BACP)

■ Parámetros:

m : Número de cursos, $m \in \mathbb{Z}^+$

n : Número de periodos académicos, $n \in \mathbb{Z}^+$

α_i : Número de créditos del curso i , $\alpha_i \in \mathbb{Z}^+$, $\forall i \in \mathbb{Z}^+ : i \in \{1, \dots, m\}$

α_i^* : Número de créditos del curso i (en binario), $\alpha_i^* \in \mathbb{Z}_2^{\lceil \log_2(\alpha_i) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$, $\forall i \in \{1, \dots, m\}$

β : Carga académica **mínima** permitida por periodo, $\beta \in \mathbb{Z}^+$

β^* : Carga académica **mínima** permitida por periodo (en binario), $\beta^* \in \mathbb{Z}_2^{\lceil \log_2(\beta) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

γ : Carga académica **máxima** permitida por periodo, $\gamma \in \mathbb{Z}^+$

γ^* : Carga académica **máxima** permitida por periodo (en binario), $\gamma^* \in \mathbb{Z}_2^{\lceil \log_2(\gamma) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

δ : Cantidad de cursos **mínima** por periodo, $\delta \in \mathbb{Z}^+$

δ^* : Cantidad de cursos **mínima** por periodo (en binario), $\delta^* \in \mathbb{Z}_2^{\lceil \log_2(\delta) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

ϵ : Cantidad de cursos **máxima** por periodo, $\epsilon \in \mathbb{Z}^+$

ϵ^* : Cantidad de cursos **máxima** por periodo (en binario), $\epsilon^* \in \mathbb{Z}_2^{\lceil \log_2(\epsilon) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

■ **Variables:**

$$X_{i,j} = \begin{cases} 1_2 & \text{si el curso } i \text{ es asignado al periodo } j, \\ & X_{i,j} \in \mathbb{Z}_2^1 : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\} \\ 0_2 & \text{en caso contrario} \end{cases}$$

c_j : carga académica del periodo j , $c_j \in \mathbb{Z}_2^{\lceil \log_2(\sum_{i=1}^m \alpha_i) \rceil} : \mathbb{Z}_2 = \{0, 1\}, \forall j \in \{1, \dots, n\}$

C : Mayor carga académica registrada a lo largo del currículum, $C \in \mathbb{Z}_2^{\lceil \log_2(\sum_{i=1}^m \alpha_i) \rceil} : \mathbb{Z}_2 = \{0, 1\}$

■ **Función objetivo:**

Minimizar la carga académica a lo largo de los periodos, lo que se puede representar como: minimizar la mayor carga académica registrada a lo largo del currículum:

Minimizar C

■ **Restricciones del problema:**

1. El dominio de $X_{i,j}$ está restringido entre 0 y 1:

$$X_{i,j} \leq 1_2,$$

$$X_{i,j} \geq 0_2,$$

$$\forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\}$$

2. La mayor carga académica C se define como:

$$C = \text{Max}\{c_1, \dots, c_n\}$$

Lo que se puede forzar usando la siguiente restricción lineal en conjunto con la función objetivo:

$$C \geq c_j, \forall j \in \{1, \dots, n\}$$

3. La carga académica para el periodo j se define como:

$$c_j = \sum_{i=1}^m \alpha_i^* \times X_{i,j}, \forall j \in \{1, \dots, n\}$$

4. Cada curso i debe ser asignados a un periodo j :

$$\sum_{j=1}^n X_{i,j} = 1_2, \forall i \in \{1, \dots, m\}$$

5. Curso b tiene como pre-requisito al curso a , por lo tanto el curso b no se puede asignar a un periodo anterior al asignado para el curso a :

$$X_{b,j} \leq \sum_{r=1}^{j-1} X_{a,r} = 1_2, \forall j \in \{2, \dots, n\}, \forall a \in \{1, \dots, m-1\}, \forall b \in \{a, \dots, m\}$$

6. La carga académica de cada periodo j debe ser mayor o igual a la carga mínima permitida:

$$c_j \geq \beta^*, \forall j \in \{1, \dots, n\}$$

7. La carga académica de cada periodo j debe ser menor o igual a la carga máxima permitida:

$$c_j \leq \gamma^*, \forall j \in \{1, \dots, n\}$$

8. El número de cursos del periodo j debe ser mayor o igual a la cantidad mínima permitida:

$$\sum_{i=1}^m X_{i,j} \geq \delta^*, \forall j \in \{1, \dots, n\}$$

9. El número de cursos del periodo j debe ser menor o igual que la cantidad máxima permitida:

$$\sum_{i=1}^m X_{i,j} \leq \epsilon^*, \forall j \in \{1, \dots, n\}$$

Nurse Scheduling Problem (NSP)

■ Parámetros:

N : Número de *nurses*, $N \in \mathbb{Z}^+$

D : Número de días que definen la jornada de trabajo, $D \in \mathbb{Z}^+$

S : Número de turnos por día (el último no turno se considera como turno de trabajo, es decir, si $S = 4$, el día contará con $S - 1 = 3$ turnos laborales), $S \in \mathbb{Z}^+$

$C_{d,s}$: Matriz de cobertura, indica la cantidad de *nurses* requerida para el día d y el turno s , $C_{d,s} \in \mathbb{Z}_0^+, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$C_{d,s}^*$: Matriz de cobertura, indica la cantidad de *nurses* requerida para el día d y el turno s (en binario), $C_{d,s}^* \in \mathbb{Z}_2^{\lceil \log_2(C_{d,s}) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$P_{n,d,s}$: Matriz de preferencias, indica la preferencia que tiene un *nurse* n para trabajar el día d en el turno s , $P_{n,d,s} \in \mathbb{Z}_0^+, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$P_{n,d,s}^*$: Matriz de preferencias, indica la preferencia que tiene un *nurse* n para trabajar el día d en el turno s (en binario), $P_{n,d,s}^* \in \mathbb{Z}_2^{\lceil \log_2(\alpha_i) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$\min_wd$: Cantidad mínima de días en que a cada *nurse* se le debe asignar un turno laboral, $\min_wd \in \mathbb{Z}_0^+$

$\min_wd^*$: Cantidad mínima de días en que a cada *nurse* se le debe asignar un turno laboral, $\min_wd^* \in \mathbb{Z}_2^{\lceil \log_2(\min_wd) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

max_wd : Cantidad máxima de días en que a cada *nurse* se le debe asignar un turno laboral (en binario), $max_wd \in \mathbb{Z}_0^+$

max_wd^* : Cantidad máxima de días en que a cada *nurse* se le debe asignar un turno laboral (en binario), $max_wd^* \in \mathbb{Z}_2^{\lceil \log_2(max_wd) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

min_cd : Cantidad mínima de días de trabajo consecutivos, $min_cd \in \mathbb{Z}_0^+$

max_cd : Cantidad máxima de días de trabajo consecutivos, $max_cd \in \mathbb{Z}_0^+$

max_cd^* : Cantidad máxima de días de trabajo consecutivos, $max_cd^* \in \mathbb{Z}_2^{\lceil \log_2(max_cd) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}$

min_cs_s : Número entero que indica la cantidad mínima de días consecutivos que un *nurse* deberá trabajar en el turno s , en caso de que se le sea asignado, $min_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

max_cs_s : Número entero que indica la cantidad máxima de días consecutivos que un *nurse* podrá trabajar en el turno s , $max_cs_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

$max_cs_s^*$: Número entero que indica la cantidad máxima de días consecutivos que un *nurse* podrá trabajar en el turno s , $max_cs_s^* \in \mathbb{Z}_2^{\lceil \log_2(max_cs_s) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, s \in \{1, \dots, S\}$

min_s_s : Número entero que indica la cantidad mínima de veces que se debe asignar el turno s a cada *nurse*, $min_s_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

$min_s_s^*$: Número entero que indica la cantidad mínima de veces que se debe asignar el turno s a cada *nurse* (en binario), $min_s_s^* \in \mathbb{Z}_2^{\lceil \log_2(min_s_s) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, s \in \{1, \dots, S\}$

max_s_s : Número entero que indica la cantidad máxima de veces que se puede asignar el turno s a cada *nurse*, $max_s_s \in \mathbb{Z}_0^+, s \in \{1, \dots, S\}$

$max_s_s^*$: Número entero que indica la cantidad máxima de veces que se puede asignar el turno s a cada *nurse* (en binario), $max_s_s^* \in \mathbb{Z}_2^{\lceil \log_2(max_s_s) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, s \in \{1, \dots, S\}$

■ **Variables:**

$$X_{n,d,s} = \begin{cases} 1_2 & \text{si el } nurse \ n \text{ es asignado al día } d \text{ en el turno } s, \\ & X_{n,d,s} \in \mathbb{Z}_2^1 : \mathbb{Z}_2 = \{0, 1\} \\ & \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\} \\ 0_2 & \text{en caso contrario} \end{cases}$$

■ **Función objetivo:**

Maximizar $H = \sum_{n=1}^N \sum_{d=1}^D \sum_{s=1}^S X_{n,d,s} \times P_{n,d,s}^*$ (*Happiness*, determinado por los turnos asignados a cada *nurse* y su preferencia sobre estos), $\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

■ **Restricciones del problema:**

1. El dominio de $X_{n,d,s}$ está restringido entre 1_2 y 0_2 :

$$X_{n,d,s} \geq 0_2,$$

$$X_{n,d,s} \leq 1_1,$$

$$\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$$

2. Cada *nurse* n debe tener asignado un turno s cada día:

$$\sum_{s=1}^S X_{n,d,s} = 1_2, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}$$

3. Cada turno laboral s en cada día d deberá tener al menos la cantidad $C_{d,s}$ de *nurses* indicada en la Matriz de cobertura:

$$\sum_{n=1}^N X_{n,d,s} \geq C_{d,s}^*, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S-1\}$$

4. A cada *nurse* se le debe asignar al menos min_wd y a lo más max_wd días de trabajo (sin contar el turno libre):

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \geq min_wd^*, \forall n \in \{1, \dots, N\}$$

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X_{n,d,s} \leq max_wd^*, \forall n \in \{1, \dots, N\}$$

5. Cada *nurse* debe tener asignado un determinado turno s al menos min_s_s y a lo más max_s_s veces:

$$\sum_{d=1}^D X_{n,d,s} \geq min_s_s^*, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

$$\sum_{d=1}^D X_{n,d,s} \leq max_s_s^*, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

6. Cada *nurse* debe trabajar al menos min_cd y a lo más max_cd días consecutivos

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} \sum_{s=1}^{S-1} X_{n,j-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i-1,s} - 2 \times \sum_{s=1}^{S-1} X_{n,i+k-1,s} \leq (k-2)_2,$$

$$\forall n \in \{1, \dots, N\}, \forall k \in \{\min\{2, min_cd\}, \dots, min_cd+1\}, \forall i \in \{1, \dots, D+2-k\}$$

$$\sum_{d=j}^{\min\{D,j+max_cd+1\}} \sum_{s=1}^{S-1} X_{n,d,s} \leq max_cd^*,$$

$$\forall n \in \{1, \dots, N\}, \forall j \in \{1, \dots, D - max_cd\}$$

7. Cada *nurse* debe repetir el mismo turno de trabajo s al menos min_cs_s y a lo más max_cs_s veces, en caso de que el turno s le sea asignado

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} X_{n,j-1,s} - 2 \times X_{n,i-1,s} - 2 \times X_{n,i+k-1,s} \leq (k-2)_2,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S-1\},$$

$$\forall k \in \{\min\{2, min_cs_s\}, \dots, min_cs_s + 1\}, \forall i \in \{1, \dots, D+2-k\}$$

$$\sum_{d=j}^{\min\{D, j+max_cs_s+1\}} X_{n,d,s} \leq max_cs_s^*,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S-1\}, \forall j \in \{1, \dots, D-max_cd\}$$

Unbounded Knapsack Problem (UKP)

■ **Parámetros:**

n : Número entero con la cantidad total de objetos, $n \in \mathbb{Z}^+$

C : Número entero con la capacidad máxima de la mochila, $C \in \mathbb{Z}^+$

W_i : Número entero indicando el peso del objeto i , $W_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

W_i^* : Número entero indicando el peso del objeto i (en binario), $W_i^* \in \mathbb{Z}_2^{\lceil \log_2(W_i) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}$

V_i : Número entero indicando el valor del objeto i , $V_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

V_i^* : Número entero indicando el valor del objeto i (en binario), $V_i^* \in \mathbb{Z}_2^{\lceil \log_2(V_i) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}$

■ **Variables:**

X_i : cantidad de veces que se agrega el objeto i a la mochila, $X_i \in \mathbb{Z}_2^{\lceil \log_2(\frac{C}{W_i}) \rceil + 1} : \mathbb{Z}_2 = \{0, 1\}, \forall i \in \{1, \dots, n\}$

■ **Función objetivo:**

Maximizar valor v en la mochila, $v = \sum_{i=1}^n V_i^* \times X_i$

■ **Restricciones del problema:**

1. Cada objeto puede ser ingresado cero o más veces:

$$X_i \geq 0, \forall i \in \{1, \dots, n\}$$

2. La suma de los pesos de los objetos ingresados no debe superar la capacidad máxima de la mochila:

$$\sum_{i=1}^n W_i^* \times X_i \leq C$$

Transformaciones de LIA a QF_ALIA

N-queens

- **Parámetros:**

N : Número de reinas que se deben posicionar en un tablero de tamaño $N \times N$, tal que $N \in \mathbb{Z}^+$

- **Variables:**

X : Arreglo de números enteros unidimensional, almacena el número de fila en E para la columna I donde se posiciona una reina, $X \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

- **Función objetivo:**

Que ninguna reina en el tablero pueda ser atacada, o en otras palabras, que se cumplan todas las restricciones del problema

- **Restricciones del problema:**

1. El dominio de $X[c]$ está restringido entre 1 y N (cantidad de filas):

$$X[c] \geq 1,$$

$$X[c] \leq N,$$

$$\forall c \in \{1, \dots, N\}$$

2. Debe haber a lo mas una reina por fila:

$$X[c_1] \neq X[c_2]$$

$$\forall c_1 \in \{1, \dots, N-1\}$$

$$\forall c_2 \in \{c_1 + 1, \dots, N\}$$

3. Debe haber a lo mas una reina por diagonal

$$X[c_2] \neq X[c_1] + (c_2 - c_1)$$

$$X[c_1] \neq X[c_2] + (c_2 - c_1)$$

$$\forall c_1 \in \{1, \dots, N-1\}$$

$$\forall c_2 \in \{c_1 + 1, \dots, N\}$$

Balanced Academic Curriculum Problem (BACP)

■ Parámetros:

m : Número de cursos, $m \in \mathbb{Z}^+$

n : Número de periodos académicos, $n \in \mathbb{Z}^+$

α_i : Número de créditos del curso i , $\alpha_i \in \mathbb{Z}^+$, $\forall i \in \mathbb{Z}^+ : i \in \{1, \dots, m\}$

β : Carga académica **mínima** permitida por periodo, $\beta \in \mathbb{Z}^+$

γ : Carga académica **máxima** permitida por periodo, $\gamma \in \mathbb{Z}^+$

δ : Cantidad de cursos **mínima** por periodo, $\delta \in \mathbb{Z}^+$

ϵ : Cantidad de cursos **máxima** por periodo, $\epsilon \in \mathbb{Z}^+$

■ Variables:

X : Arreglo de enteros bidimensional, almacena el valor 1 en E^* si se asigna el curso I al periodo I^* , almacena el valor 0 en caso contrario, $X \in \mathcal{A}_I^E, \forall I \in \mathbb{Z}, E \in \mathcal{A}_{I^*}^{E^*}, \forall I^*, E^* \in \mathbb{Z}$

c : Arreglo de números enteros unidimensional, almacena la mayor carga académica registrada en E para el periodo I , $c \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

C : Arreglo de un solo número entero, almacena la mayor carga académica registrada a lo largo del currículum E usando el índice $I = 0$, $C \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

■ Función objetivo:

Minimizar la carga académica a lo largo de los periodos, lo que se puede representar como: minimizar la mayor carga académica registrada a lo largo del currículum:

Minimizar C

■ Restricciones del problema:

1. El dominio de $X[i][j]$ está restringido entre 0 y 1:

$$X[i][j] \leq 1,$$

$$X[i][j] \geq 0,$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}$$

2. La mayor carga académica C se define como:

$$C[0] = \text{Max}\{c_1, \dots, c_n\}$$

Lo que se puede forzar usando la siguiente restricción lineal en conjunto con la función objetivo:

$$C[0] \geq c_j, \forall j \in \{1, \dots, n\}$$

3. La carga académica para el periodo j se define como:

$$c[j] = \sum_{i=1}^m \alpha_i \times X[i][j], \forall j \in \{1, \dots, n\}$$

4. Cada curso i debe ser asignados a un periodo j :

$$\sum_{j=1}^n X[i][j] = 1, \forall i \in \{1, \dots, m\}$$

5. Curso b tiene como pre-requisito al curso a , por lo tanto el curso b no se puede asignar a un periodo anterior al asignado para el curso a :

$$X[b][j] \leq \sum_{r=1}^{j-1} X[a][r] = 1, \forall j \in \{2, \dots, n\}, \forall a \in \{1, \dots, m-1\}, \forall b \in \{a, \dots, m\}$$

6. La carga académica de cada periodo j debe ser mayor o igual a la carga mínima permitida:

$$c[j] \geq \beta, \forall j \in \{1, \dots, n\}$$

7. La carga académica de cada periodo j debe ser menor o igual a la carga máxima permitida:

$$c[j] \leq \gamma, \forall j \in \{1, \dots, n\}$$

8. El número de cursos del periodo j debe ser mayor o igual a la cantidad mínima permitida:

$$\sum_{i=1}^m X[i][j] \geq \delta, \forall j \in \{1, \dots, n\}$$

9. El número de cursos del periodo j debe ser menor o igual que la cantidad máxima permitida:

$$\sum_{i=1}^m X[i][j] \leq \epsilon, \forall j \in \{1, \dots, n\}$$

Nurse Scheduling Problem (NSP)

■ **Parámetros:**

N : Número de *nurses*, $N \in \mathbb{Z}^+$

D : Número de días que definen la jornada de trabajo, $D \in \mathbb{Z}^+$

S : Número de turnos por día (el último no turno se considera como turno de trabajo, es decir, si $S = 4$, el día contará con $S - 1 = 3$ turnos laborales), $S \in \mathbb{Z}^+$

$C_{d,s}$: Matriz de cobertura, indica la cantidad de *nurses* requerida para el día d y el turno s , $C_{d,s} \in \mathbb{Z}_0^+$, $\forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

$P_{n,d,s}$: Matriz de preferencias, indica la preferencia que tiene un *nurse* n para trabajar el día d en el turno s , $P_{n,d,s} \in \mathbb{Z}_0^+$, $\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

min_wd : Cantidad mínima de días en que a cada *nurse* se le debe asignar un turno laboral, $min_wd \in \mathbb{Z}_0^+$

max_wd : Cantidad máxima de días en que a cada *nurse* se le debe asignar un turno laboral, $max_wd \in \mathbb{Z}_0^+$

min_cd : Cantidad mínima de días de trabajo consecutivos, $min_cd \in \mathbb{Z}_0^+$

max_cd : Cantidad máxima de días de trabajo consecutivos, $max_cd \in \mathbb{Z}_0^+$

min_cs_s : Número entero que indica la cantidad mínima de días consecutivos que un *nurse* deberá trabajar en el turno s , en caso de que se le sea asignado, $min_cs_s \in \mathbb{Z}_0^+$, $s \in \{1, \dots, S\}$

max_cs_s : Número entero que indica la cantidad máxima de días consecutivos que un *nurse* podrá trabajar en el turno s , $max_cs_s \in \mathbb{Z}_0^+$, $s \in \{1, \dots, S\}$

min_s_s : Número entero que indica la cantidad mínima de veces que se debe asignar el turno s a cada *nurse*, $min_s_s \in \mathbb{Z}_0^+$, $s \in \{1, \dots, S\}$

max_s_s : Número entero que indica la cantidad máxima de veces que se puede asignar el turno s a cada *nurse*, $max_s_s \in \mathbb{Z}_0^+$, $s \in \{1, \dots, S\}$

■ Variables:

X : Arreglo de enteros tridimensional, almacena el valor 1 en E^{**} para un *nurse* I que trabaja el día I^* en el turno I^{**} , almacena el valor 0 en caso contrario, $X \in \mathcal{A}_I^E, \forall I \in \mathbb{Z}, E \in \mathcal{A}_{I^*}^{E^*}, \forall I^* \in \mathbb{Z}, E^* \in \mathcal{A}_{I^{**}}^{E^{**}}, \forall I^{**}, E^{**} \in \mathbb{Z}$

■ Función objetivo:

Maximizar $H = \sum_{n=1}^N \sum_{d=1}^D \sum_{s=1}^S X_{n,d,s} \times P_{n,d,s}$ (*Happiness*, determinado por los turnos asignados a cada *nurse* y su preferencia sobre estos), $\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$

■ Restricciones del problema:

1. El dominio de $X[n][d][s]$ está restringido entre 1 y 0:

$$X[n][d][s] \geq 0,$$

$$X[n][d][s] \leq 1,$$

$$\forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S\}$$

2. Cada *nurse* n debe tener asignado un turno s cada día:

$$\sum_{s=1}^S X[n][d][s] = 1, \forall n \in \{1, \dots, N\}, \forall d \in \{1, \dots, D\}$$

3. Cada turno laboral s en cada día d deberá tener al menos la cantidad $C_{d,s}$ de *nurses* indicada en la Matriz de cobertura:

$$\sum_{n=1}^N X[n][d][s] \geq C_{d,s}, \forall d \in \{1, \dots, D\}, \forall s \in \{1, \dots, S-1\}$$

4. A cada nurse se le debe asignar al menos min_wd y a lo más max_wd días de trabajo (sin contar el turno libre):

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X[n][d][s] \geq min_wd, \forall n \in \{1, \dots, N\}$$

$$\sum_{d=1}^D \sum_{s=1}^{S-1} X[n][d][s] \leq max_wd, \forall n \in \{1, \dots, N\}$$

5. Cada *nurse* debe tener asignado un determinado turno s al menos min_s_s y a lo más max_s_s veces:

$$\sum_{d=1}^D X[n][d][s] \geq min_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

$$\sum_{d=1}^D X[n][d][s] \leq max_s_s, \forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S\}$$

6. Cada *nurse* debe trabajar al menos min_cd y a lo más max_cd días consecutivos

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} \sum_{s=1}^{S-1} X_{n,j-1,s} - 2 \times \sum_{s=1}^{S-1} X[n][i-1][s] - 2 \times \sum_{s=1}^{S-1} X[n][i+k][s] \leq k-2,$$

$$\forall n \in \{1, \dots, N\}, \forall k \in \{\min\{2, min_cd\}, \dots, min_cd+1\}, \forall i \in \{1, \dots, D+2-k\}$$

$$\sum_{d=j}^{\min\{D,j+max_cd+1\}} \sum_{s=1}^{S-1} X[n][d][s] \leq max_cd,$$

$$\forall n \in \{1, \dots, N\}, \forall j \in \{1, \dots, D - max_cd\}$$

7. Cada *nurse* debe repetir el mismo turno de trabajo s al menos min_cs_s y a lo más max_cs_s veces, en caso de que el turno s le sea asignado

$$\sum_{j=\max\{i+1,2\}}^{\min\{i+k+1,D\}} X[n][j-1][s] - 2 \times X[n][i-1][s] - 2 \times X[n][i+k-1][s] \leq k-2,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S-1\},$$

$$\forall k \in \{\min\{2, \min_cs_s\}, \dots, \min_cs_s + 1\}, \forall i \in \{1, \dots, D + 2 - k\}$$

$$\sum_{d=j}^{\min\{D, j+\max_cs_s+1\}} X[n][d][s] \leq \max_cs_s,$$

$$\forall n \in \{1, \dots, N\}, \forall s \in \{1, \dots, S - 1\}, \forall j \in \{1, \dots, D - \max_cd\}$$

Unbounded Knapsack Problem (UKP)

■ **Parámetros:**

n : Número entero con la cantidad total de objetos, $n \in \mathbb{Z}^+$

C : Número entero con la capacidad máxima de la mochila, $C \in \mathbb{Z}^+$

W_i : Número entero indicando el peso del objeto i , $W_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

V_i : Número entero indicando el valor del objeto i , $V_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$

■ **Variables:**

X : Arreglo de números enteros unidimensional, almacena la cantidad de veces E que se agrega el objeto I en la mochila, $X \in \mathcal{A}_I^E, \forall I, E \in \mathbb{Z}$

■ **Función objetivo:**

Maximizar valor v en la mochila, $v = \sum_{i=1}^n V_i \times X[i]$

■ **Restricciones del problema:**

1. Cada objeto puede ser ingresado cero o más veces:

$$X[i] \geq 0, \forall i \in \{1, \dots, n\}$$

2. La suma de los pesos de los objetos ingresados no debe superar la capacidad máxima de la mochila:

$$\sum_{i=1}^n W_i \times X[i] \leq C$$

Tablas con información adicional recopilada

En esta sección del anexo se incluyen las tablas usadas para construir las figuras mostradas en el Capítulo 4.

Instancias (n =)	LIA	LRA	QF_BV	QF_ALIA
4	0.0	0.0	0.01	0.02
8	0.01	0.01	0.01	0.01
12	0.01	0.04	0.03	0.04
16	0.01	0.73	0.04	0.09
20	0.01	0.54	0.08	0.31
50	53.48	945.57	1.15	94.22
100	12.67	86391.36	27.62	35633.14
120	6.34	86396.01	17.28	83088.13
140	21.74	86389.97	15.89	86399.08
160	413.68	86387.12	11.04	86398.97
180	12.21	88649.89	38.15	86398.41
200	91.41	88642.09	50.92	86395.83

Tabla 5.1: Tabla con el tiempo total (en segundos) que duró la búsqueda de soluciones de cada uno de los modelos planteados en el Capítulo 3 para el problema *N-queens*, usada para elaborar la Figura 4.1. Los mejores tiempos se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Instancias	LIA	LRA	QF_BV	QF_ALIA
25_1_g00	49.48	38.86	134.42	7.47
50_1_g00	0.73	102.44	578.14	1448.87
75_1_g00	13.2	239.82	2504.73	3586.69
100_1_g00	484.9	85850.05	7549.23	72025.12
30_1_g00	86629.77	85862.48	16735.05	86416.2
60_1_g00	69395.79	86416.38	86397.08	86420.51

Tabla 5.2: Tabla con el tiempo total (en segundos) que duró la búsqueda de soluciones de cada uno de los modelos planteados en el Capítulo 3 para el problema *NSP*, usada para elaborar la Figura 4.2. Los mejores tiempos se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Instancias	LIA	LRA	QF_BV	QF_ALIA
n10c7370	4.41	0.88	36666.44	0.21
n20c17050	41.67	1.82	84387.09	1.71
n30c27795	293.62	4.62	84378.31	3.37
n40c39588	4451.47	15.24	84374.65	37.95
n50c41934	19092.61	17.83	84368.93	64.06
n60c57601	81853.96	48.19	85225.78	223.94
n70c68135	84610.13	43.94	84619.23	251.39
n80c67603	59136.41	92.3	84623.47	260.78

Tabla 5.3: Tabla con el tiempo total (en segundos) que duró la búsqueda de soluciones de cada uno de los modelos planteados en el Capítulo 3 para el problema *UKP*, usada para elaborar la Figura 4.3. Los mejores tiempos se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Instancias	LIA	LRA	QF_BV	QF_ALIA
five	0.02	0.04	0.1	0.04
p01	86418.61	2444.65	164.43	16076.5
gr17	86396.28	86402.67	8723.9	87117.35
fri26	86404.37	86414.56	86411.71	87161.1
att48	86507.92	86493.32	86495.12	87199.18

Tabla 5.4: Tabla con el tiempo total (en segundos) que duró la búsqueda de soluciones de cada uno de los modelos planteados en el Capítulo 3 para el problema *TSP*, usada para elaborar la Figura 4.4. Los mejores tiempos se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Instancia	LIA	LRA	QF_BV	QF_ALIA
bacp8	3.64	86520.45	86511.48	20.66
bacp10	1900.91	86476.59	86464.46	12.58
bacp12	86406.43	10.79	86395.37	88000.44

Tabla 5.5: Tabla con el tiempo total (en segundos) que duró la búsqueda de soluciones de cada uno de los modelos planteados en el Capítulo 3 para el problema *BACP*, usada para elaborar la Figura 4.5. Los mejores tiempos se distinguen por tener letra más oscura. Fuente: Elaboración propia.

Bibliografía

- [1] R. Oliver, “Optimization modulo theories,” Master’s thesis, Universitat Politècnica de Catalunya, 01 2011.
- [2] K. Aksoy, S. Tahar, and Y. Zeren, “Introduction to hol4 theorem prover,” *Sigma J Eng Nat Sci*, vol. 10, 2019.
- [3] M. Bofill, J. Coll, J. Suy, and M. Villaret, “Solving the multi-mode resource-constrained project scheduling problem with smt,” in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 239–246, 2016.
- [4] J. Rintanen, “Discretization of temporal models with application to planning with smt,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, Mar. 2015.
- [5] M. Eraqsu, F. Micota, and D. Zaharie, “Scalable optimal deployment in the cloud of component-based applications using optimization modulo theory, mathematical programming and symmetry breaking,” *Journal of Logical and Algebraic Methods in Programming*, vol. 121, p. 100664, 2021.
- [6] E. Davidson, Ö. Akgün, J. Espasa, and P. Nightingale, “Effective encodings of constraint programming models to smt,” in *Principles and Practice of Constraint Programming* (H. Simonis, ed.), (Cham), pp. 143–159, Springer International Publishing, 2020.
- [7] N. Gálvez Ramírez, “Modelos de programación lineal.” Universidad Técnica Federico Santa María. University Lecture, 2019.
- [8] B. S. Hussain, “CSPLib problem 054: N-queens.” <http://www.csplib.org/Problems/prob054>.
- [9] M. Al-Rudaini, “N-queens problem solving using linear programming in gnu linear programming kit (glpk),” 11 2016.

- [10] S. A. Cook, “The complexity of theorem-proving procedures,” *Proceedings of the third annual ACM symposium on Theory of computing*, vol. STOC ’71, p. 151–158, 1971. Archived (PDF) from the original on 2022-10-09.
- [11] A. Biere and D. Kröning, *SAT-Based Model Checking*, pp. 277–303. Cham: Springer International Publishing, 2018.
- [12] M. Abdulaziz and F. Kurz, “Formally verified sat-based ai planning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 14665–14673, Jun. 2023.
- [13] S. Eggergluss, G. Fey, and R. Drechsler, “Sat-based atpg for path delay faults in sequential circuits,” in *2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3671–3674, 2007.
- [14] V. Nicolet, “Csc410 tutorial: z3 - sat solving.” <http://www.cs.toronto.edu/~victor-n/tutorials/z3/SAT.html>, 2017. Last accessed on 15-10-2023.
- [15] L. de Moura and N. Bjørner, “Z3: an efficient smt solver,” vol. 4963, pp. 337–340, 04 2008.
- [16] C. Tinelli, “A dpll-based calculus for ground satisfiability modulo theories,” in *Logics in Artificial Intelligence* (S. Flesca, S. Greco, G. Ianni, and N. Leone, eds.), (Berlin, Heidelberg), pp. 308–319, Springer Berlin Heidelberg, 2002.
- [17] L. de Moura and N. Bjørner, “Satisfiability modulo theories: An appetizer,” in *Formal Methods: Foundations and Applications* (M. V. M. Oliveira and J. Woodcock, eds.), (Berlin, Heidelberg), pp. 23–36, Springer Berlin Heidelberg, 2009.
- [18] C. Barrett, P. Fontaine, and C. Tinelli, “The SMT-LIB Standard: Version 2.6,” 2017. Available at www.SMT-LIB.org.
- [19] L. De Moura and N. Bjørner, “Satisfiability modulo theories: Introduction and applications,” *Commun. ACM*, vol. 54, p. 69–77, sep 2011.
- [20] N. Gálvez Ramírez, *A Framework for Autonomous Generation of Strategies in Satisfiability Modulo Theories*. PhD thesis, 12 2018.
- [21] M. Bofill, J. Suy, M. Palahí, and M. Villaret, “Solving constraint satisfaction problems with sat modulo theories,” *Constraints*, vol. 17, pp. 273–303, 2012.
- [22] M. Bofill, J. Suy, and M. Villaret, “A system for solving constraint satisfaction problems with smt,” in *Theory and Applications of Satisfiability Testing – SAT 2010* (O. Strichman and S. Szeider, eds.), (Berlin, Heidelberg), pp. 300–305, Springer Berlin Heidelberg, 2010.
- [23] M. Palahí i Sitges, *Reformulation of constraint models into SMT*. PhD thesis, Spain, 2015.

- [24] H. Hosobe, “Solving hierarchical soft constraints with an smt solver,” in *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, ICCAE 2020, (New York, NY, USA), p. 42–46, Association for Computing Machinery, 2020.
- [25] H. Kim, *Efficient Smt Solving for Hardware Model Checking*. PhD thesis, USA, 2010. AAI3433303.
- [26] A. Armando, J. Mantovani, and L. Platania, “Bounded model checking of software using smt solvers instead of sat solvers,” in *Model Checking Software* (A. Valmari, ed.), (Berlin, Heidelberg), pp. 146–162, Springer Berlin Heidelberg, 2006.
- [27] N. Bjørner, “Smt in verification, modeling, and testing at microsoft,” in *Hardware and Software: Verification and Testing* (A. Biere, A. Nahir, and T. Vos, eds.), (Berlin, Heidelberg), pp. 3–3, Springer Berlin Heidelberg, 2013.
- [28] L. Cordeiro, B. Fischer, and J. Marques-Silva, “Smt-based bounded model checking for embedded ansi-c software,” in *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 137–148, 2009.
- [29] C. Ansótegui, M. Bofill, F. Manyà, and M. Villaret, “Automated theorem provers for multiple-valued logics with satisfiability modulo theory solvers,” *Fuzzy Sets and Systems*, vol. 292, pp. 32–48, 2016. Special Issue in Honor of Francesc Esteva on the Occasion of his 70th Birthday.
- [30] C. Ansótegui, M. Bofill, F. Manyà, and M. Villaret, “Building automated theorem provers for infinitely-valued logics with satisfiability modulo theory solvers,” in *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, pp. 25–30, 2012.
- [31] S. Böhme, *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.
- [32] K. R. M. Leino, “Automating theorem proving with smt,” in *Interactive Theorem Proving* (S. Blazy, C. Paulin-Mohring, and D. Pichardie, eds.), (Berlin, Heidelberg), pp. 2–16, Springer Berlin Heidelberg, 2013.
- [33] B. Dutertre and L. de Moura, “A fast linear-arithmetic solver for dpll(t),” in *Computer Aided Verification* (T. Ball and R. B. Jones, eds.), (Berlin, Heidelberg), pp. 81–94, Springer Berlin Heidelberg, 2006.
- [34] C. Barrett, L. de Moura, and A. Stump, “Smt-comp: Satisfiability modulo theories competition,” in *Computer Aided Verification* (K. Etessami and S. K. Rajamani, eds.), (Berlin, Heidelberg), pp. 20–23, Springer Berlin Heidelberg, 2005.
- [35] R. Sebastiani and S. Tomasi, “Optimization in smt with la(q) cost functions,” 2012.

- [36] R. Sebastiani and P. Trentin, “Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions,” in *Tools and Algorithms for the Construction and Analysis of Systems* (C. Baier and C. Tinelli, eds.), (Berlin, Heidelberg), pp. 335–349, Springer Berlin Heidelberg, 2015.
- [37] P. Trentin and R. Sebastiani, “Optimization modulo the theory of floating-point numbers,” in *Automated Deduction – CADE 27* (P. Fontaine, ed.), (Cham), pp. 550–567, Springer International Publishing, 2019.
- [38] N. Bjørner, A. Phan, and L. Fleckenstein, “vz - an optimizing smt solver,” pp. 194–199, 04 2015.
- [39] R. Sebastiani and P. Trentin, “Optimathsat: A tool for optimization modulo theories,” pp. 447–454, 07 2015.
- [40] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, “cvc5: A versatile and industrial-strength smt solver,” in *Tools and Algorithms for the Construction and Analysis of Systems* (D. Fisman and G. Rosu, eds.), (Cham), pp. 415–442, Springer International Publishing, 2022.
- [41] N. S. Bjørner and A.-D. Phan, “vz - maximal satisfaction with z3,” in *International Symposium on Symbolic Computation in Software Science*, 2014.
- [42] N. Gálvez Ramírez, Y. Hamadi, E. Monfroy, and F. Saubion, “Evolving smt strategies,” in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 247–254, 11 2016.
- [43] N. G. Ramírez, E. Monfroy, F. Saubion, and C. Castro, “Optimizing smt solving strategies by learning with an evolutionary process,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 816–820, 2018.
- [44] N. Gálvez Ramírez, Y. Hamadi, E. Monfroy, and F. Saubion, “Towards automated strategies in satisfiability modulo theory,” in *European Conference on Genetic Programming*, pp. 230–245, 03 2016.
- [45] N. Gálvez Ramírez, E. Monfroy, F. Saubion, and C. Castro, “Improving complex smt strategies with learning,” *International Transactions in Operational Research*, vol. 27, no. 2, pp. 1162–1188, 2020.
- [46] Q. Z. Benjamin Mikek, “Speeding up smt solving via compiler optimization,” in *In Proceedings of the Symposium on Foundations of Software Engineering (FSE)*., 2023.
- [47] J. Schmidt and M. Leuschel, “Improving smt solver integrations for the validation of b and event-b models,” in *Formal Methods for Industrial Critical Systems* (A. Lluch Lafuente and A. Mavridou, eds.), (Cham), pp. 107–125, Springer International Publishing, 2021.

- [48] Lu, Zhengyang, “Alphasmt: A reinforcement learning guided smt solver,” Master’s thesis, 2023.
- [49] M. Balunovic, P. Bielik, and M. Vechev, “Learning to solve smt formulas,” in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.
- [50] R. Sebastiani and P. Trentin, “On optimization modulo theories, maxsmt and sorting networks,” 02 2017.
- [51] T. Gawlitza and D. Monniaux, “Improving strategies via SMT solving,” *CoRR*, vol. abs/1101.2812, 2011.
- [52] I. P. Gent, C. Jefferson, and P. Nightingale, “Complexity of n-queens completion,” *Journal of Artificial Intelligence Research*, vol. 59, pp. 815–848, 2017.
- [53] B. Hnich, Z. Kiziltan, and T. Walsh, “CSPLib problem 030: Balanced academic curriculum problem (bacp).” <http://www.csplib.org/Problems/prob030>.
- [54] M. Chiarandini, L. D. Gaspero, S. Gualandi, and A. Schaerf, “The balanced academic curriculum problem revisited,” *Journal of Heuristics*, vol. 18, p. 119–148, 2012.
- [55] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition ed., 1979.
- [56] R. Lagatie, S. Haspeslagh, and P. D. Causmaecker, “Negotiation protocols for distributed nurse rostering.” Eindhoven University of Technology Department of Computer Science, 2009.
- [57] S. J. den Hartog, H. Hoogeveen, and T. C. van der Zanden, “On the complexity of nurse rostering problems,” *Operations Research Letters*, vol. 51, no. 5, pp. 483–487, 2023.
- [58] J. S. Arora, “16 - genetic algorithms for optimum design,” in *Introduction to Optimum Design (Second Edition)* (J. S. Arora, ed.), pp. 531–542, San Diego: Academic Press, second edition ed., 2004.
- [59] W. J. Cook, B. Korte, L. Lovász, A. Wigderson, and G. M. Ziegler, eds., *The Traveling Salesman Problem*, pp. 527–562. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
- [60] Özgür Akgün, “CSPLib problem 133: Knapsack problem.” <http://www.csplib.org/Problems/prob133>.
- [61] T. C. Hu, L. Landa, and M.-T. Shing, *The Unbounded Knapsack Problem*, pp. 201–217. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

- [62] C. Castro and S. Manzano, “Variable and value ordering when solving balanced academic curriculum problems,” 2001.
- [63] M. Vanhoucke and B. Maenhout, “Nsplib—a nurse scheduling problem library: a tool to evaluate (meta-) heuristic procedures,” 01 2007.
- [64] J. Burkardt, “Data for the traveling salesperson problem.” <http://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html>, 2011.
- [65] G. Reinelt, “TSPLIB—a traveling salesman problem library,” *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.
- [66] H. Becker and L. S. Buriol, “An empirical analysis of exact algorithms for the unbounded knapsack problem,” *European Journal of Operational Research*, vol. 277, no. 1, pp. 84–99, 2019.
- [67] C. E. Miller, A. W. Tucker, and R. A. Zemlin, “Integer programming formulation of traveling salesman problems,” *J. ACM*, vol. 7, p. 326–329, oct 1960.
- [68] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli, “Towards smt model checking of array-based systems,” in *Automated Reasoning* (A. Armando, P. Baumgartner, and G. Dowek, eds.), (Berlin, Heidelberg), pp. 67–82, Springer Berlin Heidelberg, 2008.
- [69] N. Bjørner, L. Moura, L. Nachmanson, and C. Wintersteiger, *Programming Z3*, pp. 148–201. 04 2019.