**UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA**

**INSTITUTO DE INGENIERÍA**

**MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA**


AREA DE CONOCIMIENTO COMPUTACIÓN

# PLANIFICACIÓN DE TAREAS PARA LA ETAPA DE GRABADO EN SISTEMAS DE MANUFACTURA DE SEMICONDUCTORES MEDIANTE UN ENFOQUE HÍBRIDO BASADO EN ANSWER SET PROGRAMMING Y PROGRAMACIÓN CON RESTRICCIONES

TESIS

que para obtener el grado de

DOCTORADO EN CIENCIAS

presenta

**CARMEN LETICIA GARCIA MATA**

DIRECTORA

**Dra. Larysa Burtseva**

CODIRECTOR

**Dr. Víctor Hugo Yaurima Basaldúa**

Mexicali, B.C 2016

**RESUMEN** de la Tesis de **Carmen Leticia García Mata**, presentada como requisito parcial para la obtención de grado de Doctorado en Ciencias, Mexicali, Baja California, Octubre 14 del 2016.

**PLANIFICACIÓN DE TAREAS PARA LA ETAPA DE GRABADO DE SISTEMAS DE MANUFACTURA DE SEMICONDUCTORES MEDIANTE UN ENFOQUE HÍBRIDO BASADO EN ANSWER SET PROGRAMMING Y PROGRAMACIÓN CON RESTRICCIONES**

Resumen aprobado por : _____ _____

Dra Larysa Burtseva — Directora de Tesis

Dr. Víctor Hugo Yaurima Basaldúa — Codirector de Tesis

La planificación de tareas y la optimización de planes ha sido un tema central de investigación en el área de computación. En particular, el cada vez más demandante y sofisticado mercado de los sistemas de manufactura de semiconductores, requiere cada vez más y mejores planes. La simplicidad del planteamiento teórico de un problema de planificación de tareas, no permite entrever a primera vista, como la solución de este problema se torna rápidamente intratable conforme la entrada crece. Desde hace más de medio siglo, se han propuesto numerosas teorías y técnicas provenientes desde áreas tan dispares como Investigación de Operaciones, Razonamiento Automatizado y Cómputo Inteligente para la solución de este tipo de problemas. Algunas de las técnicas más eficientes son las máquinas de inferencia genéricas basadas tanto en lógica proposicional como en otras lógicas, incluidas la lógica monotónica. Las máquina más exitosos, están basadas en DPPL y su extensión CDCL y pueden resolver problemas con miles de variables y millones de reglas.

En esta tesis se explora el uso de metodologías y motores de inferencia basados en lógica no monotónica para la solución de problemas de planificación de tareas en sistemas de manufactura. Para demostrar su aplicabilidad en problemas de planificación, se presenta el modelado y solución de un problema de planificacion de tareas y optimización del plan para un Sistema Automatizado de Grabado (AWS) de una empresa de semiconductores. El modelado del problema y su solución se realizan con un enfoque híbrido basado en Answer Set Programming y Programación con Restricciones, ASP modulo CSP. Los resultados obtenidos, en su mayoría, son comparativamente mejores que con los paradigmas MILP y CSP sin ASP.

Palabras clave: Answer Set Programming, Planificación de Tareas, Problemas Combinatorios, Razonamiento No-Monotónico, Programación con Restricciones, Sistemas de Manufactura

**ABSTRACT** of the thesis, presented by CARMEN LETICIA GARCÍA MATA, in order to obtain the DOCTOR IN SCIENCE DEGREE in COMPUTER SCIENCE. Mexicali, Baja California, México, October 14, 2016.

**SCHEDULING FOR THE WET-ETCH STAGE SEMICONDUCTOR MANUFACTURING SYSTEMS USING A HYBRID APPROACH BASED ON ANSWER SET PROGRAMMING AND CONSTRAINT PROGRAMMING**

**Abstract**

Scheduling and Optimization has been a central research theme in the area of computing. In particular, the increasingly demanding and sophisticated market for semiconductor manufacturing systems, requires increasingly better plans. The simplicity of the theoretical approach of most scheduling problems, does not allow at first glance that the solution of this problem quickly becomes intractable as the input grows. For over half a century, researchers around the world have proposed numerous theories and techniques derived from so disparate areas such as Operations Research, Automated Reasoning and Soft Compuing to solving this kind of problem. Some of the most efficient techniques are generic inference machines based on both, propositional logic and other logic, including monotonic logic. The ones most successful are those based in the DPPL/CDDL algorithms, and can solve problems with thousands of variables and millions of rules.

This thesis investigates the applicability of methodologies and inference engines based on non-monotonic logic to solve scheduling and optimization problems in manufacturing systems through solution of a practical problem in semiconductor industry. The application consisted in modeling and solving a scheduling and optimization problem for an Automated Wet-etch System (AWS) of a semiconductor company with an hybrid approach based on Answer Set Programming and Constraint Programming approach (ASP module CSP). The results, in most cases, are comparatively better than the MILP paradigm and the CSP paradigm.

# Índice

**Índice de Figuras**

**Índice de Tablas**

# Capítulo 1

## INTRODUCCIÓN

### 1.1 Motivación

El valor del mercado global de la industria de manufactura de semiconductores en 2015 fue de $332 billones de dólares (equivalentes a $6 billones 424,200 millones de pesos mexicanos, al tipo de cambio 19.35 del 12 de sept. Del 2016). Además, se tienen firmes expectativas de que continúe creciendo rápidamente los próximos cinco años, puesto que el uso de electrónicos permea cada vez más en todos los ámbitos de nuestra vida (reportsnreports.com). La enorme demanda de estos dispositivos, deviene en una presión inmensa para los fabricantes de semiconductores para que incrementen la variedad de productos y reduzcan sus tiempos de entrega.

En este tipo de empresas, las órdenes de producciones pueden variar bruscamente desde órdenes que demandan un alto volumen de productos no muy variados, hasta órdenes en los que se requiere un volumen bajo de una inmensa variedad de productos. Esta variación en el tipo de órdenes hace que el problema de producción de semiconductores sea extremadamente complejo y obliga al diseño de plantas de manufactura flexibles.

Las fuerzas del mercado y lo complejo del proceso de manufactura han provocado que la industria se esté moviendo rápidamente hacia sistemas de manufactura automatizados para resolver el problema de acortar los tiempos de entrega del producto y reducir los costos de manufactura.

Una parte esencial de estos sistemas automatizados es el planificador del proceso de producción. Frecuentemente, el planificador consta de al menos dos módulos que operan de manera jerárquica. Uno de estos módulos es denominado el planificador general y es el que establece *qué* tareas ejecutar. Dependiendo el grado de integración del sistema automatizado, el planificador general integra en su base de conocimiento información relativa a los distintos niveles de producción, como por ejemplo: cadena de productores, almacén, ventas, etc. El otro módulo del planificador es el planificador de tareas (*scheduler*), y su función consiste en especificar *cómo* ejecutar las tareas encomendadas, usando un número limitado de recursos y una cantidad limitada de tiempo.

El tema del diseño de un planificador general queda fuera del ámbito de interés de esta

investigación, por ello, nos concentraremos exclusivamente en el planificador de tareas. Aunque efectivamente el diseño de un planificador general es un problema de mayor complejidad, no por ello el diseño del planificador de tareas implica un esfuerzo trivial. A continuación se detallará un breve desarrollo histórico en el que se da cuenta acerca de la complejidad del problema, los enfoques, teorías y técnicas tejidas alrededor de este problema y otros similares que comparten características en común y que inclusive han generado el desarrollo de nuevos campos de investigación.

La investigación de planificación de tareas puede ser trazada hasta inicios del siglo XX con el trabajo seminal de Gantt quién creó innovadoras gráficas de control de producción y creó también muchas otras gráficas que daban una visión del sistema de manufactura y medían diferentes variables que afectan el desempeño de los sistemas de manufactura.

Durante los años 40 del siglo pasado Dantzig inventó el algoritmo *simplex*, el cual es una técnica general y extremadamente poderosa para resolver problemas de programación lineal y es considerado como probablemente el mejor algoritmo que haya sido inventado puesto que resulta aplicable a múltiples y complejos programas de planificación y optimización. Es también gracias a este algoritmo que se considera a Dantzig como el padre fundador de la Programación Lineal. Para un estudio más detallado acerca del algoritmo *simplex*, remitimos al lector al artículo fundamental de Programación Lineal en el cual Dantzig (1955) discute cuestiones acerca de la solución de problemas díficiles de programación lineal e implementación de su destacado algoritmo.

La importancia de la investigación de Dantzig puede ser ponderada desde dos puntos de vista: 1) Desde el punto de vista de planificación y optimización, puesto que su algoritmo *simplex* y variantes de éste son utilizadas actualmente por múltiples lenguajes de modelado y sistemas de software, y consistentemente resuelven problemas conteniendo miles de variables y restricciones 2) Desde el punto de vista de teoría de complejidad, porque también en esta área Dantzig realizó trabajo pionero que fue muy útil para el posterior desarrollo del área de complejidad, ya que propuso un buen número de técnicas para expresar varios tipos de restricciones complejas como sistemas de desigualdades lineales, restringiéndolas además a valores enteros (Dantizig, 1960).

Sin embargo, no fue sino hasta finales de los años 50 y principios de los años 60 del siglo anterior, con el adviento de las primeras computadores, que se formalizó la investigación en esta área. Por cuestiones de brevedad, solo resaltaremos los logros más importantes que sentaron las bases teóricas de planificación de tareas. En el área de planificación de tareas, se desarrollaron varios

algoritmos para problemas de secuenciamiento, como por ejemplo, la regla de Johnson para el problema de taller de máquina, la regla para minimizar la máxima latencia (*earliest due date*, EDD) y la regla para el procesamiento del tiempo más corto (*Shortest Processing Time*, SPT) y la regla para minimizar el tiempo de flujo promedio (Herrmann, 2006).

También durante los años 50 del siglo pasado, se diseñaron veloces algoritmos para resolver diferentes problemas de optimización, tales como diseño de circuitos para computar funciones booleanas, problemas de teoría de grafos y el clásico, el problema del vendedor (*Traveling Sales Person*, TSP), que básicamente consiste en encontrar la trayectoria Hamiltoniana más corta en un grafo de vértices ponderados.

Pronto fue notado que aunque todos estos problemas son teóricamente solubles, puesto que solo tenían que explorar un espacio de solución finito, en la práctica son intratables porque conforme el tamaño de los datos de entrada crece, el espacio de solución crece también rápidamente de manera exponencial.

Desde entonces, la noción complejidad de tiempo fue definida como la posibilidad de "explosión combinatoria", lo que conlleva a una ineficiencia extrema y la imposibilidad práctica de utilizar algoritmos de búsqueda de fuerza bruta para la solución de estos problemas.

Fueron precisamente los problemas combinatorios los que de manera crucial lograron que los investigadores desplazaran su atención desde cómputo efectivo hacia cómputo adecuado. Fueron también los problemas combinatorios los que marcaron la diferencia entre algoritmos eficientes e ineficientes. Para más detalles acerca del desarrollo teórico de ciencias computacionales se aconseja la lectura de los artículos de Karp (1986), Garey & Johnson(1979) y Johnson (2012).

Hartmanis & Stearn(1965) formalizaron el concepto de complejidad de tiempo y además notaron que los mejores algoritmos para la solución de difíciles problemas combinatorios tenían también complejidad de tiempo exponencial.

Posterior a los trabajos de Hartmanis & Stearn, la comunidad científica de computación continuó trabajando arduamente buscando un algoritmo general que resolviese problemas con complejidad exponencial. Este optimismo declinó bruscamente después que Cook (1971) definió formalmente en su trabajo seminal, las clases de lenguajes P y NP y demostró que el problema SAT es NP-complete. En este trabajo, Cook estableció también la cuestión: ¿es P=NP o es P≠NP?. La respuesta aún no ha sido encontrada hasta la fecha, pero sigue siendo motivo de enorme investigación porque tiene

profundas implicaciones en muchos campos de aplicación, especialmente en el tema de seguridad informática.

Otra investigación de relevancia relacionada con este tema de investigación, es la realizada por Karp(1972), quien demostró que las versiones de decisión de muchos problemas de optimización bien conocidos pertenecen a la clase complejidad NP-complete.

Las implicaciones de los trabajos de Cook (1971) y Karp(1972) resultan relevantes para todos los problemas combinatorios, en especial para el problema de satisfabilidad, SAT, y los problemas de decisión, porque implican, que a menos que P=NP, no existe un algoritmo eficiente para la solución de estos problemas.

Estos dos trabajos tuvieron un impacto dual en la comunidad de investigación de ciencias computacionales. Por un lado, ciertos grupos de investigación dirigieron su atención hacia el área de Teoría de Complejidad, buscando comprender la estructura de los problemas NP. Otros grupos de investigación se dieron a la búsqueda, no ya del algoritmo más eficiente, sino a la búsqueda de algoritmos aproximados que resultaran convenientes para aplicaciones prácticas.

Por otra parte, otro grupo de investigadores del área de razonamiento automatizado buscaban concretar el sueño de construir máquinas de razonamiento automático. Aunque los primeros intentos pueden trazarse hasta Leibniz, la primera vez que se estableció como un problema formal fue debido a Hilbert con su planteamiento conocido como el problema *Entscheidungs* (Hilbert & Ackerman, 1928). El problema plantea si es posible encontrar un procedimiento que dada cualquier expresión lógica permita decidir mediante un número finito de operaciones su validez o satisfabilidad.

La respuesta al problema de Hillbert fue negativa y fue demostrada por Turing (1936) mediante el problema de interrupción (*halting problem*); en tanto Church(1936) lo demostró mediante el cálculo lambda. No obstante, si el conjunto de modelos se restringe a satisfacer solamente ciertas teorías específicas, el problema es decidible.

Uno de los métodos que resultó exitoso para construir pruebas de teoremas, es el principio de resolución de Robinson (1965), el cual construye las pruebas mediante refutación. El método resultó convenientemente adecuado para construir probadores automáticos de teoremas. Sin embargo, tiene un inconveniente: la resolución siempre confirma o refuta una sentencia, pero no obtiene conocimiento nuevo y solo aplica a disjunciones de literales (Luckham, 1967). Estas características hacen que el

método de resolución sólo sea aplicable a bases de conocimiento y preguntas conformadas por disjunciones.

Otra problema interesante que finalmente llevó al desarrollo de una de las máquina más exitosas de inferencias para la resolución automatizada, es el problema de satisfacibilidad, más conocido como SAT. El objetivo en este problema es que dada una sentencia o un conjunto de sentencias en su forma conjuntiva normal y utilizando un algoritmo de búsqueda con retroceso (*backtracking*), determinar si la sentencia es satisfacible o no. El problema es resuelto por el algoritmo conocido como DPPL, cuyo nombre proviene de los investigadores que diseñaron el algoritmo: Davis, Putnam, Logemann y Loveland (Davis & Putnam, 1960) (Davis et al., 1962). El algoritmo de búsqueda es un procedimiento de ramificación sistemático y completo que poda el espacio de búsqueda en base a cláusulas falsificadas.

A poco más de 50 años de su creación, DPPL y sus versiones extendidas, CDCL (*Conflict-Driven Clause Learning*) (Marques-Silva & Sakallah, 1999), son los algoritmos básicos que conforma la mayoría de los modernos motores de búsqueda basados en SAT. La eficiencia de los actuales resolvedores SAT recae en extensiones y mejoras que se han agregado durante todas estas décadas, entre las cuales se incluye la unidad de propagación, los mecanismos de aprendizaje, estrategias de reinicio aleatorias y determinísticas, mecanismos para borrar cláusulas y heurísticas de búsqueda inteligente.

Aunque los resolvedores SAT son altamente eficientes en la solución de muchos problemas combinatorios relacionados con problemas prácticos de muchas áreas, también tienen una deficiencia: puesto que están basados en lógica monotónica, no son adecuados para representar conocimiento dinámico y contradictorio.

En una línea de investigación relacionada, surgió la investigación de diversas lógicas, comúnmente agrupadas bajo el término de lógicas no monotónicas. Las primeras investigaciones relevantes al tema surgieron a mitad del siglo pasado, y son atribuidas a McCarthy(1959), pero no fue hasta finales de los años 80 de ese mismo siglo, que se logró desarrollar una semántica lo suficientemente simple y eficiente para ser automatizada. Esta teoría es la Semántica del Modelo Estable (*Stable Model Semantic, SME*), es autoría de Gelfond & Lifschitz(1988) y supuso un punto de inflexión a partir del cual la investigación en el área creció enormemente. Unos pocos años después fue identificada como un nuevo paradigma al cual se le denominó Answer Set Programming (ASP).

Desde entonces se han desarrollado numerosos y eficientes motores de inferencia para ASP, tales como Smodels, DLV, Cmodels, Pbmodels, Clasp y DLV-Hex.

ASP tiene aplicabilidad en multitud de áreas, como por ejemplo planificación, ruteo, satisfacción de restricciones, robótica, diagnóstico, análisis de seguridad y un largo etcétera. Ello gracias a que la fecha existen numerosas extensiones e integraciones de ASP con otros paradigmas como por ejemplo Problemas con Satisfacción de Restricciones (*Constraint Satisfaction Problem, CSP*), lógica difusa, logica de descripciones (*Description Logic, DL*), satisfacibilidad de Teorías de Módulo (*Satisfiability Modulo Theories, SMT*).

SMT surge en el campo de SAT como una generalización de este último, y también como un método para combinar SAT con otros paradigmas. Posteriormente fue adaptada en la comunidad de ASP. Recientemente ha sido propuesto un nuevo lenguaje llamado ASPMT (Bartholomew & Lee, 2014), basado también en la Semántica del Modelo Estable. ASPMT es, a su vez, una generalización de ASP. La máquina de inferencias de ASPMT, la cual hasta la fecha es un prototipo basada en una serie de resolvedores: el compilador de ASPMT, f2lp, gringo y Z3, es un lenguaje declarativo de alto nivel que permite cómputo eficiente con números reales y generación de planes con cambios continuos.

En resumidas cuentas, la automatización de sistemas de manufactura para fábricas de semiconductores y en particular el desarrollo de planificadores y *schedulers* para su proceso de producción representan un reto teórico y tecnológico formidable. Afortunadamente, actualmente existe un amplio abanico de métodos para resolución de problemas complejos. La metodología basada en sistemas de inferencia parece especialmente bien situada para resolver este tipo de problemas.

## 1.2 Planteamiento del Problema

En esta tesis se presenta el modelado y solución de un problema de planificación de tareas y la optimización del plan resultante para una Estación de Grabado de Obleas (*Automated Wet-Etch Station, AWS*) de una empresa de semiconductores. El problema de planificación consiste en determinar la secuencia de lote de obleas en cada baño. La función objetivo es la minimización del *makespan (*donde *makespan* es el tiempo que se tardan en ejecutar los procesos en todas las máquinas a las son asignado). De acuerdo al proceso de producción, todos los trabajos deben ser procesados secuencialmente en todas las máquinas, iniciando siempre en la primer máquina y terminando en la

última. Los trabajos son ininterrumpibles. Las máquinas representan un conjunto de baños químicos y baños de agua dispuestos alternadamente. El plan resultante tiene que cumplir con restricciones estrictas en el sentido de que una vez que un trabajo termine su procesamiento en un baño químico, inmediatamente después debe ser asignado al siguiente baño de agua. Los baños de agua fungen también como almacenamiento temporal hasta que se libere el siguiente baño químico. Se asume que los baños no requieren de ningún tipo de reinicio (*setup*) y que no hay rupturas asociadas con éstos.

Aunque en sistemas reales de manufactura, los lotes son movidos entre los baños por uno ó más robots, en esta investigación se simplifica el problema, considerando que la velocidad del robot es constante y los tiempos de traslado son tan pequeños en relación con la duración de los procesos, que es despreciable y no tomada en cuenta. Además, dado que los tiempos de operación de estos robots son del orden de milisegundos, y el lenguaje ASP, con el cual se resolverá el problema, solo opera con números enteros, por lo tanto no es posible diseñar programas ASP que incorporen información que requiera el manejo de números reales.

Dadas las características y restricciones, el problema de planificación consiste en determinar la secuencia de lote de obleas en cada baño. La meta es la minimización del *makespan*. Además, el problema puede ser descrito como un Problema de Planificación de Estaciones de Taller de Flujo Seriales Multiproducto con Políticas ZW/NIS y LS/NIS

Puesto que los problemas de planificación de tareas son motivo de investigación científica al menos desde los años 40 de la anterior centuria, existen múltiples métodos para el modelado y solución del problema. Entre los más empleados, podemos citar los métodos exactos utilizados en Investigación de Operaciones, tales como los enumerativos, de ramificación y acotamiento (*branch and bound*), programación dinámica, programación entera y lineal (*linear and integer programming*), etc. Dentro de los métodos no exactos están los métodos heurísticos y los metaheurísticos (*lógica difusa, redes neuronales, algoritmos genéticos, etc.*). Alternativamente, existen otros métodos para la solución de problemas de planificación de tareas provenientes del área de razonamiento automatizado. Estos métodos se puede clasificar en: a) basados en lógica monotónica (ej. SAT, ) y los que están basados en lógica no monotónica como ASP.

Para la solución del problema de planificación y optimización, seleccionamos las metodologías no monotónicas, específicamente el lenguaje ASP y extensiones y/o integraciones que sean necesarias, en conjunto con el motor de inferencias Clingo. La selección de este lenguaje resulta conveniente

porque el modelo matemático puede ser representado casi directamente con ASP, gracias a la alta expresividad del lenguaje. Su base teórica monotónica, facilita el razonamiento con conocimiento incompleto y contradictorio. Además, considerando que actualmente existe una tendencia al diseño de sistemas de manufactura basados en el conocimiento, resulta estratégico adelantarse al futuro e iniciar desde ahora la construcción de los planificadores (que son la parte esencial de cualquier sistema de manufactura automatizada) mediante técnicas basadas en el conocimiento.

## 1.3 Objetivos

### 1.3.1 Objetivo General

Investigar el uso de ASP en el diseño, implementación y solución de problemas de planificación. En particular, el objetivo es modelar y resolver un problema de planificación de tareas y optimizar el plan resultante para una AWS de una empresa de semiconductores.

### 1.3.2 Objetivos Específicos

**i.** Realizar la optimización del plan (*schedule*) mediante la minimización del *makespan*.

**ii.** Medir la capacidad del motor de inferencias Clingo en la solución de problemas con explosión combinatoria, como lo son los problemas de planificación.

**iii.** Analizar y seleccionar algún otro de los módulos que conforman la serie de motores de inferencia híbridos de ASP para modelar y resolver el problema con un enfoque ASP híbrido, como por ejemplo el módulo que permite integrar Programación con Restricciones y ASP.

**iv.** Comparar los resultados obtenidos con ASP y ASP híbrido con respecto a los resultados obtenidos con los paradigmas de Programación con Restricciones y MILP publicados por Zeballos et al. (2011) y Bhushan & Karimi (2003).

## 1.4 Metodología

Se realizó una búsqueda, selección y análisis de literatura interdisciplinaria a fin de identificar la metodología que se está empleando actualmente para la

solución de problemas de *scheduling* en sistemas de manufactura. Puesto que el objetivo de esta investigación también esta dirigido a identificar, seleccionar y analizar metodologías y tecnologías emergentes que resulten adecuadas para resolver el problema en cuestión, también se realizó una búsqueda bibliográfica y revisión del estado del arte en el área de Representación del Conocimiento y Razonamiento (*Knowledge Representation and Reasoning, KRR*), en particular, la búsqueda se dirigió hacia la subárea de Razonamiento Automático y solución de problemas combinatorios complejos.

La selección de estas áreas de estudio están fundadas en el hecho de que hay una fuerte tendencia al desarrollo de Sistemas de Manufactura Avanzados basadas en el conocimiento y es necesario identificar tecnologías prometedoras que permitan desarrollar este tipo de sistemas.

La instancia del problema seleccionado se basó en datos de un sistema del tipo AWS de una fábrica de semiconductores publicado por Zeballos et al. (2012). La estrategia para la selección de esa instancia en particular del problema se fundó en que dicho investigador resolvió el problema con un método proveniente también del área de Razonamiento Automático.

Una vez identificada la metodología y tecnologías emergentes más adecuadas para la solución del problema de scheduling seleccionado, se procedió a seleccionar la máquina de inferencias que se utilizaría para encontrar los modelos o soluciones. Durante esta etapa se instaló y probó la operación de algunas máquinas de inferencias mediante la solución de diversos ejemplos.

Ya con la selección de la máquina de inferencias, se procedió a modelar, representar y resolver el problema en cuestión mediante aproximaciones, incrementado progresivamente el número de trabajos y el número de máquinas. Se siguió este método en la solución del problema, debido a que al tratarse de un problema combinatorio, conforme se incrementa el número de variables, el número de

posibles combinaciones crece exponencialmente. Entonces, el modelo diseñado para un problema menor tamaño, no funciona igual cuando el tamaño del problema se incrementa, ya que el tiempo necesario para la solución de un problema con más variables, crece enormemente, y para efectos prácticos, el tiempo necesario para encontrar la solución óptima es inadmisible. Entonces, se fue refinando el modelo, delimitándolo para reducir el espacio de búsqueda y encontrar una solución en tiempo polinomial.

Uno de los enfoques que se siguió para refinar el problema fue el de generar el espacio de búsqueda mediante rangos. Es decir, en lugar de generar las combinaciones para cada baño independientemente de su posición dentro de la secuencia ordenada de procesamiento, se explotó precisamente esta característica para generar el rango de combinaciones para cada baño dependiendo de la posición que le corresponda dentro de la secuencia. Este enfoque permitió reducir hasta en un 99.5% el porcentaje de combinaciones, permitiendo resolver problemas con un mayor número de trabajos y baños.

Pero esta modificación no fue suficiente para resolver las instancias del problema que se había planteado inicialmente, se procedió entonces, a analizar que otras extensiones o integraciones de paradigmas a ASP tenían las cualidades que nos permitirían resolver el problema planteado. Como resultado de este análisis se seleccionó el paradigma ASP modelo CSP, el cual es una combinación de ASP y Programación con Restricciones. La utilización de un enfoque híbrido como el propuesto, resulta conveniente para la solución del problema tema de esta investigación, porque este tipo de problemas son NP-hard y tienen un espacio de búsqueda enorme. Por otro lado, el cuello de botella de ASP es la etapa de instanciación, especialmente para problemas con datos masivos. En cambio, bajo el paradigma CSP es posible resolver problemas con variables globales y están optimizados para trabajar con datos masivos.

Una vez seleccionado el paradigma y resolvedor híbrido ASP + CSP, se procedió a modelar e implementar el problema bajo este nuevo enfoque.

## I.5 Esquema General de la Tesis

**Capítulo 1.** En este capítulo se presenta la motivación que dio origen a esta tesis. También se hace el planteamiento del problema, los objetivos generales y específicos y por último se detalla sucintamente cual fue la metodología empleada para la solución del problema

**Capítulo 2.** Describe las características de los problemas combinatorios, y la influencia que el estudio de estos problemas ha tenido en el desarrollo teórico de computación en general, y en particular cómo la búsqueda a la solución de este tipo de problemas ha impulsado la creación de diversas máquinas de inferencias basadas en distinto tipo de máquinas.

**Capítulo 3.** Estudia los diferentes modelos de producción para ambientes estáticos y se plantea cómo denotar y clasificar este tipo de problemas. En este capítulo se analiza la complejidad de los algoritmos de planificación de tareas y optimización.

**Capítulo 4.** Describe las taxonomías que existen para la descripción de problemas de replanificación. Se propone una combinación de dos de estas taxonomías y en base a esta clasificación, se presenta un análisis del estado del arte sobre replanificación.

**Capítulo 5.** Se presenta la descripción formal del problema de planificación de tareas para un AWS de un Sistema de Manufactura de Semiconductores y cómo se tradujo este modelo en el lenguaje declarativo para representación del conocimiento, ASP.

**Capítulo 6.** Se discute a profundidad los resultados computacionales obtenidos, describiendo las diferentes aproximaciones que se hicieron al problema utilizando diferentes estrategias y enfoques. Se presenta también una comparación de los resultados obtenidos en esta investigación con los resultados obtenidos por otros investigadores que utilizaron la metodología de Programación con Restricciones y MILP.

**Capítulo 7.** Se discute con más amplitud las estrategias que se siguieron para resolver el problema, cuáles fueron las dificultades que se tuvieron para la solución del problema y cómo se solventaron éstas. Se justifica la razón por la cual fue necesario resolver el problema con un enfoque híbrido

basado en ASP+CSP.

**Capítulo 8.** Plantea diversas líneas de acción que se pueden seguir para extender la solución del problema de planificación de tareas resuelto, tanto desde el punto de vista conceptual como desde el punto de vista de utilizar metodologías de reciente creación, proponiendo específicamente cuáles serían estas tecnologías y por qué sería conveniente su utilización.

## Capítulo 2

## SISTEMAS BASADOS EN EL CONOCIMIENTO

Las nuevas tendencias en el diseño e implementación de sistemas avanzados de manufactura se basan en un modelado estructurado, cuya piedra angular es el conocimiento. En este enfoque se utilizan frecuentemente metodologías que permitan analizar, estructurar y crear bases de conocimiento y sistemas de razonamiento automático (Borgo & Leitao, 2004), (Negri et al., 2015).

Las tecnologías basadas en conocimiento son cada vez más necesarias para resolver problemas en áreas muy dispares. Actualmente muchas grandes industrias están considerando el uso de instrumentos semánticos en el modelado de problemas sobre dominios complejos para resolver problemas del mundo real. Como una respuesta a esta necesidad, recientemente se han desarrollado un conjunto de poderosos métodos y técnicas del área de representación del conocimiento y razonamiento (*Knowledge*

*Representation and Reasoning*, KRR) (Harmelen et al., 2008), (Davis & Marcus, 2015).

En este capítulo se discutirán brevemente los antecedentes, ventajas y limitaciones de lógica clásica para el diseño de razonadores automatizados. La discusión se extenderá al análisis de lógica no monotónica y bases teóricas de los modernos razonadores automatizados basados tanto en lógica clásica como en lógica no monotónica.

## 2.1 Delimitación del Área de AI y Retos en su Aplicabilidad

En la literatura no existe una definición de inteligencia que sea aceptada de manera unánime. En cambio, la mayoría de los investigadores del área de inteligencia artificial están de acuerdo en que una de las características que debe reunir una entidad inteligente es la capacidad de adaptabilidad y la capacidad de resolver los problemas nuevos que se le presenten (Kanazawa, 2004). Es decir, se acepta que la inteligencia de un ente está relacionada directamente con la adaptabilidad del individuo al medio ambiente. Relacionar la inteligencia con la adaptabilidad tiene sentido, puesto que adaptarse implica percibir, analizar información, sacar conclusiones, planificar y actuar en consecuencia.

Desde el punto de vista de inteligencia artificial hay dos definiciones de inteligencia: a) inteligencia general (*General or Strong AI*) , y b) inteligencia débil (*Weak AI*) (Proudfoot, 2011), (Goertzel, B., 2007). Mientras que los investigadores en el área de inteligencia débil solo pretenden desarrollar entidades o sistemas capaces de comportarse "inteligentemente", los investigadores del área de inteligencia general tienen pretensiones mucho más altas ya que buscan construir "entes inteligentes" que no solo se comporten "inteligentemente", sino que posean creencias propias, creatividad y conciencia, y por lo tanto sean capaces de reflexionar sobre sus propias acciones, sus sentimientos y aún sobre sus mismos pensamientos (McDermott, 2007),(Steunbrink & Schmidhuber, 2012) .

No existe una fecha precisa de cuándo se logrará el objetivo de inteligencia general, pero probablemente se lleven muchas décadas para que esto ocurra, si es que ocurre. En cambio, los logros del área de inteligencia débil ó inteligencia computacional son ya evidentes en muchos de los dispositivos y servicios inteligentes (Forrest & Hoanca, 2015).

Por otra parte, aunque durante décadas se ha trabajado en el diseño de sistemas automatizados de manufactura, las soluciones tradicionales son a menudo rígidas. Por ejemplo, el control en sistemas de

tiempo real está basado en información fragmentada y dispersa y una estructura de hardware rígida (Shipt et al, 2012). Este tipo de diseños es estable solo en ambientes estáticos.

Para administrar un sistema de producción de este tipo, los directivos requieren de una entrada externa (por ejemplo, información acerca del producto y la orden de trabajo). Generalmente la información respecto de la logística de la empresa, los algoritmos de planificación y los datos de configuración se almacenan en un sistema de bases de datos externa (Gunnman & Eneyo, 2016). Cada sistema es construido de manera independiente, lo que complica aún más la compartición de información entre los distintos niveles jerárquicos.

Aunque en ocasiones, una parte de la información requerida se programa directamente en los dispositivos encargados de manejar el equipo (*Devices Control Units*, DCUs). Dependiendo de la forma que se haya solucionado este punto en un sistema dado, los DCUs son incrustadas en las unidades que controlan la manufactura (*Manufacturing Control Units*, MCUs), los controladores del cómputo númerico (*Computer Numeric Controls*, CNCs) o de los controladores lógicos programables (*Programmable Logic Controllers*, PLCs) (Diedrich et al., 2015).

## 2.2 Problemas de Búsqueda Combinatoria: KRR y ASP

### 2.2.1 Clasificación de Problemas Combinatorios

Los problemas combinatorios permean prácticamente todas las áreas de investigación. La característica distintiva de esta clase de problemas es que el número de posibles soluciones es finito pero crece exponencialmente conforme se incrementa el tamaño de la entrada. Algunos problemas combinatorios típicos son: búsqueda de trayectorias más cortas/baratas, ruteo de paquetes de datos en internet, predicción de estructura de proteínas, planificación y calendarización de recursos.

En teoría, es posible encontrar la solución óptima de un problema combinatorio dado realizando una búsqueda exhaustiva de todas las posibles combinaciones de las variables que conforman el problema. Desafortunadamente, aunque el número de posibles combinaciones es finito, también es extremadamente grande. En la mayoría de los casos, encontrar una solución óptima que satisfaga un cierto conjunto de condiciones previamente establecido, es un problema intratable, aún para problemas

de tamaño relativamente modesto.

En la teoría de complejidad, los problemas combinatorios se dividen por clases, y algunas de las más relevantes son: optimización combinatoria (*Combinatorial Optimization*), satisfabilidad (*Satisfiability,* SAT), el problema de satisfacción de restricciones (*Constraint Satisfaction Problem,* CSP), satisfacción de fórmulas booleanas cuantificadas (*Quantified Boolean Formulas*, QBF), planificación (de longitud limitada), planificación de tareas (*Scheduling*). El espacio de búsqueda de soluciones, usualmente es exponencial en el tamaño de la entrada y su complejidad es del tipo **NP**-completo (*NP-complete*) para todos estos problemas, excepto para el caso de QBF, el cual es **PSPACE**-completo (Garey & Johnson, 1979), (Samulowitz, 2008).

Actualmente no existe un método general que nos permita lidiar con toda clase de problemas combinatorios. Comúnmente, se ha adoptado un enfoque híbrido para la solución de este tipo de problemas con diferentes técnicas basadas en metaheurísticas que van desde la inteligencia artificial (*Artificial Intelligence*, AI) hasta la investigación de operaciones (Jourdan et al., 2009)(Mingers & White, 2010). De entre las técnicas metaheurísticas basadas en AI, destacadamente se emplean las redes neuronales (García-Pedrajas et al., 2006), la lógica difusa (Zhao et al., 2010) y los algoritmos bioinspirados (Mohan & Baskaran, 2012). Sin embargo, dada la complejidad del problema, usualmente se emplea una combinación de heurísticas y métodos de búsqueda combinatoria para obtener una solución en tiempo razonable (Blum et al., 2011).

Recientemente se han desarrollado varias tecnologías de razonamiento automático. Estas tecnologías están basadas principalmente en lógica proposicional (*Propositional Logic*, LP) (Pham et al., 2009), (Hölldobler et al., 2014), (Bruynooghe et al., 2015), pero también se han desarrollado nuevas teorías y técnicas con lógicas no-monotónicas (Gelfond & Lifschitz, 1988), (Pelov et al., 2007), (Shen & Eiter, 2016). Es importante conocer, cuáles son las características de estas tecnologías para determinar cual es la más conveniente para resolver cada clase de problemas combinatorios. En los siguientes apartados nos enfocamos especialmente en determinar aquellas que resulten más útiles para resolver problemas de la planificación en ambientes de manufactura.

### 2.2.2 Razonadores SAT y LP

La subárea de razonamiento automático de AI, tiene como principal objetivo la resolución de este tipo de problemas mediante el desarrollo de teorías, técnicas y lenguajes para la representación de

problemas que son resueltos mediante razonadores (*solvers*) automáticos (Harrison, 2009). SAT fue una de las primeras máquinas de inferencias en obtener buenos resultados en la resolución de problemas prácticos. SAT está basado en la LP. Aún más, no solo SAT está basado en LP sino muchas otros razonadores modernos se basan también en LP (Hölldobler et al., 2014).

Esto parece un tanto extraño si tomamos en cuenta que LP es poco expresivo y que la representación de problemas del mundo real en LP requiere una inmensa cantidad de cláusulas, más allá de lo deseable para fines prácticos. Sin embargo, LP tiene en su favor que es **decidible** y mucho más conveniente computacionalmente que otras lógicas de orden más alto, como por ejemplo la lógica de primer orden, la cual ha sido demostrado que es indecidible (Kieroński & Otto, 2005). A continuación, analizaremos a más detalle las características sintácticas, semánticas y método de resolución utilizando LP.

**Sintaxis de LP**

Las sentencias atómicas consisten de un solo símbolos proposicional, por ejemplo, P, Q, R. La sintaxis de LP se representa como sigue:

*i.* Cada símbolo significa una proposición.

*ii.* Los símbolos sólo se valúan a *True* o *False* (Verdadero o Falso).

*iii.* Se usan mayúsculas para los símbolos.

*iv.* Las sentencias complejas se forman usando los conectivos lógicos permitidos: $\neg$ , $\wedge$ , $\vee$ , $\Rightarrow$ **,** que significan **not**, **and**, **or** y **entonces**, respectivamente.

*v.* La gramática formal de LP en notación booleana BNF (*Backus-Naur Form*) es (Knuth, 1964).

*Sentencia* $\rightarrow$ *Sentencia Atómica* | *Sentencia Compleja*

*Sentencia Compleja* $\rightarrow$ *True* | *False* | *Símbolo*

*Símbolo* $\rightarrow$ *P* | *Q* | *R* | ..

*Sentencia Compleja* $\rightarrow$ $\neg$ *Sentencia*

| (*Sentencia* $\wedge$ *Sentencia*)

| (*Sentencia* $\vee$ *Sentencia*)

| (*Sentencia* $\Rightarrow$ *Sentencia*)

| (*Sentencia* $\Leftrightarrow$ *Sentencia*)

**Semántica de LP:** Especifica cómo determinar el valor de verdad de cada sentencia en un modelo *m.*

*i.* El valor de verdad de cada sentencia atómica está dado por *m.*

*ii.* El valor de verdad de cualquier otra sentencia es obtenido recursivamente usando tablas de verdad.

Para decidir si una serie de cláusulas es verdadera, se sigue un razonamiento que permite determinar si las expresiones están lógicamente vinculadas entre sí. En notación matemática, estaría expresado por $\alpha \models \beta$, lo que significa que $\beta$ es verdadero *si y solo si* $\alpha$ también es verdadero (Newell at al., 1963).

En la figura 2.1 se muestra la tabla de verdad para cada uno de los operadores válidos en LP. Dadas las variables booleanas *a* y *b*, se infiere si alguna proposición como la siguiente cláusula es verdadera o falsa:

$$(a \wedge b) \vee (\neg a) \vee (\neg b) = ?$$

| **a** | **b** | **¬a** | **a ∧ b** | **a ∨ b** | **a ⇒ b** |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

Figura 2.1 Tabla de verdad para operadores básicos de LP

Si suponemos que $a = 0$ y $b = 1$, entonces la cláusula toma el siguiente valor:

$(0 \wedge 1) \vee (\neg 0) \vee (\neg 1) = 0 \vee 1 \vee 0 = 1.$

El razonamiento lógico se realiza de dos formas: manualmente mediante tablas de verdad, ó de manera automática mediante algún algoritmo. Los algoritmos generales hacen una enumeración

recursiva del espacio finito de asignación de las variables de las cláusulas, y tienen como característica principal que son **lógicamente válidos** y **completos**. Un método de inferencia es **válido** si todas las conclusiones que obtiene son verdaderas, y **completo** si cumple con dos requisitos: a) obtiene cualquier conclusión derivable de la base de conocimiento, y b) siempre termina (teóricamente plausible, ya que el número de modelos es finito) (Plümper et al., 2011). La complejidad de un algoritmo general es $O(2^n)$. Sucintamente, esto significa que si, en conjunto, la base de conocimiento (BC*)* y la conclusión $\alpha$, contienen *n* símbolos, entonces existen $2^n$ combinaciones (modelos). En el peor caso, ése sería el número de combinaciones que se requiere analizar para encontrar una solución.

Aunque hay algoritmos más eficientes que los generales, todos los algoritmos conocidos para LP tienen una complejidad exponencial, con respecto al tamaño de la entrada, en el peor caso. Esto no es extraño ya que LP tiene complejidad co-**NP**-completo (Delgrande & Gupta,1993).

La figura 2.2 muestra el número estimado de variables/reglas que requerirían varios problemas en diferentes dominios si el conocimiento necesario para resolver un problema fuese descrito con LP. También se aprecia cómo se relacionan estos problemas con las clases de complejidad.

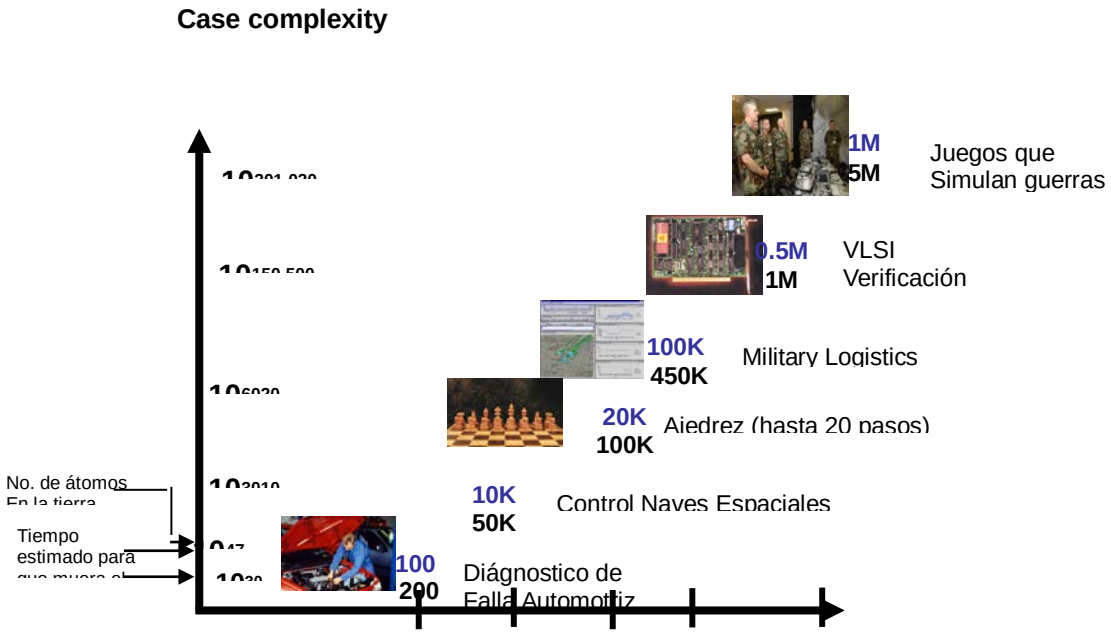


Figura 2.2 Número estimado de cláusulas para la solución de distintos problemas mediante LP.

El problema de la explosión combinatoria hace que el algoritmo general sea poco útil para la solución de problemas reales mediante razonamiento automatizado. Para reducir el espacio de búsqueda se crearon métodos de resolución más efectivos, pero buscando siempre que éstos sean válidos y completos (Dechter & Rish, 1994). El método de resolución de Robinson (1965), fue el primero que reunía estas características.

A fin de reducir el tamaño y el tiempo de búsqueda, fueron diseñados nuevos algoritmos como el algoritmo DPPL y sus variantes (Gómes et al., 2008). El objetivo de estos algoritmos es responder al cuestionamiento de si cierta cláusula es satisfacible en algún modelo.

**El Problema SAT**

El problema de satisfabilidad proposicional (SAT) es: Dado un conjunto de cláusulas sobre un conjunto *U* finito. Las cláusulas deben estar en la Forma Conjuntiva Normal (*Conjuctive Normal Form*, CNF) ; encontrar una asignación para *U* que satisfaga todas las cláusulas (Kautz & Selman, 2007).

Una cláusula está en su forma conjuntiva si y solo si las cláusulas se relacionan con operadores AND y los términos con operadores OR, por ejemplo:

$$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3).$$

Los primeros algoritmos para solucionar SAT fueron construidos desde los 60's, por los mismos investigadores que crearon el demostrador automático de teoremas (de siglas ATP, por el término en inglés: *Automated Theorem Proving*). Luego, estos algoritmos fueron mejorados por David & Putnam (1960) y después por Loveland & Logemann (1962), integrando estas mejoras en un algoritmo conocido ampliamente como DPLL. Desafortunadamente este algoritmo solo resulta conveniente para resolver problemas pequeños, mientras que las instancias de problemas reales requieren millones de variables y cláusulas.

A partir de los años 90 del siglo anterior, el método DPLL fue mejorado significativamente cuando se desarrolló un nuevo algoritmo que constituye la base de la mayoría de las máquinas de inferencias SAT modernas. Este algoritmo se llamó aprendizaje de cláusulas manejadas por conflicto (*Conflict Drive Clause Learning*, CDCL) (Marques & Sakallah,1999). CDCL está basado en las

heurísticas VSIDS, análisis de conflicto y dos estructuras más eficientes conocidas como “listas de literales dos veces observadas”.

CDCL a su vez, ha sido mejorado en las últimas décadas, de tal forma que las máquinas de inferencias SAT ahora son capaces de resolver en unos pocos segundos problemas con cientos o miles de variables y millones de cláusulas (Hölldobler et al., 2014). Una descripción a *grosso modo* del algoritmo CDCL, se muestra en el siguiente párrafo.

**Algoritmo de Búsqueda CDCL de SAT**

- Busca en el espacio de soluciones mediante una estrategia de primero en profundidad.
- Unidad de Propagación: si $a$ es verdadero, entonces $\neg a$ no satisface una cláusula.
- Entonces, $\neg a \vee b \vee \neg c$ se convierte en $b \vee \neg c$ .
- Continúa el mismo procedimiento hasta que encuentra un conflicto (fin de camino).
- Extrae una cláusula aprendida.
- Salta hacia atrás algunos niveles y propaga la cláusula aprendida.
- Aplica heurísticas Adaptivas para Poda (sopesa las literales en conflicto).
- Y después, reinicia, simplifica la base de cláusulas, elimina cláusulas, etc.

Las aplicaciones más importantes de SAT son las relacionadas a problemas industriales en el diseño automatizado electrónico (*Electronic Design Automated*, EDA). En este campo, los problemas más importantes, en los cuáles se aplica SAT para su solución, son (Marques & Sakallah, 2000):

- Verificación de equivalencia (*Equivalence Checking)*
- Verificación del modelo (*Model Checking*).
- Generación automática del patrón de prueba (*Automatic Test Pattern Generation*, ATPG).
- Síntesis lógica (por ejemplo, ruteo FPGA e identificación de redundancia).

Sin embargo, las aplicaciones de SAT no se restringen a EDA. Otra área de aplicación importante del SAT es la de Verificación de Software. Recientemente se empieza a explorar la aplicabilidad del SAT a otros problemas combinatorios como la generación de horarios (*Timetabling*), la planificación con un solo recurso y restricciones relajadas entre trabajos (*cumulative scheduling*) y la planificación de eventos deportivos (*Sports Scheduling*) (Abío, 2013).

La aplicación del SAT ha sido remarcablemente exitosa en muy diversos problemas. Gran parte

de su éxito es debido al gran número de eficientes máquinas de inferencias que han sido desarrolladas la última década . Otro factor que también ha influido al éxito de SAT es debido a que muchos problemas reales pueden ser resueltos en tiempo polinomial puesto que experimentalmente se ha encontrado que tales problemas presentan estructuras del tipo estático y/ó dinámico, las cuáles pueden ser aprovechadas por la máquina de inferencias para encontrar la solución en un lapso de tiempo práctico. Para más detalles sobre este tema, vea (Gómes et al., 2000), (Gaspers et al., 2014).

Por otro lado, se ha determinado que los problemas de planificación mediante representación simbólica pertenecen a la clase de problemas del tipo **PSPACE**-*complete*, pero si un problema se restringe a planes de tamaño polinomial, entonces es soluble en tiempo polinomial y se resuelva eficientemente en una máquina de inferencias SAT (Kautz & Selman, 1996).

Si un problema de planificación no es acotado, no es resoluble con SAT. Por cuanto hace a los problemas relativos a planificación de tareas, muchos de estos problemas son **NP**-duros (**NP**-*hard*) y tampoco se resuelven directamente por el SAT, a menos que sea posible transformarlo a un problema SAT.

## 2.3 Programación con Restricciones

La idea básica de la programación con restricciones (*Constraint Programming*, CP) es definir un problema como un conjunto de restricciones representadas a través de un conjunto de variables. Se define un dominio de valores para cada variable y también se establecen las relaciones entre los subconjuntos de estas variables. Por ejemplo, en un problema de planificación de tareas para un sistema de manufactura, las variables de decisión son los tamaños de los lotes de cada trabajo y los tiempos de inicio de las trabajos. En tanto, las restricciones posibles son: la capacidad de los recursos (por ejemplo, un trabajo se procesa a la vez en cada máquina), y la forma en que los trabajos son procesados (por ejemplo, los trabajos no deben ser procesados en un orden arbitrario, sino que deben cumplir con las especificaciones de las órdenes.

Recientemente, CP ha sido extendido para diferentes propósitos. Algunas de estas extensiones fueron diseñadas para la solución de problemas de optimización con uno ó múltiples criterios. Los razonadores de CP (*CP solver*s), hacen la búsqueda de la solución de una forma local o sistemática. Los

razonadores basados en búsqueda sistemática usan resolución de búsqueda hacia atrás (*backtracking*), ramificación y acotamiento (*branch and bound*), o una combinación de búsqueda e inferencia. La inferencia es usada para angostar el espacio de búsqueda. Sin embargo, muchos problemas combinatorios no se reducen en tiempo polinomial y por tanto es impráctico realizar máquinas que hacen búsqueda sistemática. Es por esta causa, que aún es común el uso de razonadores basados en búsqueda no-sistemática ó búsqueda local, aún y cuando las soluciones devueltas no siempre son óptimas (Goultiaeva &Bacchus, 2012).

Al igual que SAT, el CSP es **NP**-completo, afortunadamente, es frecuente que las instancias de problemas reales de optimización combinatoria muestran algunas propiedades estructurales que se explotan para diseñar algoritmos que encuentren la solución en tiempo polinomial. Algunas de estas propiedades estructurales son estáticas, mientras que otras son dinámicas.

Entre las propiedades estáticas, está el grado de aciclicidad de los grafos de restricciones. Por ejemplo, una instancia de problemas de optimización combinatoria es soluble en tiempo polinomial si el ancho del árbol (*tree-width*) de su grafo de restricciones está acotado por una constante (Samer & Szeider, 2010). Por otra parte, el enfoque basado en propiedades estructurales dinámicas tiene como estrategia la búsqueda de estructuras escondidas, las que se determinan en tiempo de ejecución mediante el análisis de los datos generados durante la instanciación. Estas estructuras son conocidas como puertas traseras (*backdoor set*) relativas a las variables, cuya instanciación transforma un problema **NP**-completo a uno tratable (Dilkina et al., 2007),(Gómes et al., 2008), y (Gaspers et al., 2014).

Si bien, los razonadores de CP han sido aplicados exitosamente en la solución de problemas reales, también tienen algunos inconvenientes. Uno de ellos es que frecuentemente los problemas reales se encuentran sobre-restringidos y no existe una solución para ese conjunto dado de restricciones. Una forma en que este problema ha sido resuelto, es usar preferencias en lugar de restricciones, o modificar ligeramente las restricciones de forma que se encuentre una solución sin modificar demasiado el problema original (Downing et al, 2013) y (Hoeve, 2011).

A pesar de estos inconvenientes, se han reportado casos de éxito en la resolución de problemas de planificación/replanificación para sistemas de manufactura mediante CP, como se ve en los reportes de investigación (Lombardi & Milano, 2012), (Zeballos et al., 2011). Otro caso, donde CP fue aplicada para resolver un problema de planificación de tareas en SMS, está dado en (Novas & Henning, 2010).

## 2.4 Lógica No-monotónica y ASP

Dentro de la counidad de investigadores de KRR existen grupos que encaran el problema de la construcción de sistemas inteligentes desde enfoques muy diferentes. Hay grupos que dirigen su investigación hacia cuestiones fundamentales comunes a diferentes aplicaciones, y consecuentemente, al desarrollo de métodos generales. Otros grupos tienen el objetivo de desarrollar métodos especializados de KRR para lidiar con dominios que forman el núcleo de los razonadores, tales como tiempo, espacio, causación y acción. Un tercer grupo de investigadores ha puesto su atención en la solución de aplicaciones prácticas mediante técnicas de KRR. Consecuentemente, la planificación se encuentra dentro de esta clase de aplicaciones.

Los métodos generales propuestos hasta hoy tienen puntos en común, pero difieren en cuestiones importantes. Una de las diferencias más importantes se debe al tipo de lógica que se usa, especialmente si es la lógica monotónica ó la lógica no-monotónica. La lógica monotónica es mayormente conocida como la lógica clásica. Desafortunadamente, la lógica clásica es suficientemente inexpresiva e impráctica para lidiar con conocimiento incompleto y contradictorio (Hanks & McDermott, 1986). En cambio, los humanos hacemos todo el tiempo inferencias de ese tipo mediante una estrategia de "normalmente" (*normally)*. Por ejemplo, cuando nos dirigimos a nuestras actividades diarias, seleccionamos la trayectoria que nos lleve al destino en el menor tiempo posible, basándose en un conocimiento de "normalmente": "Normalmente tomo la avenida *X* y a la altura de *Y* volteo a la izquierda, tomando por la avenida *Z*, y desde *Z* conduzco directamente 10 minutos hasta arribar al sitio deseado".

Esta estrategia nos desliga de la necesidad de conocer diariamente todos los posibles imprevistos que nos podrían impedir llegar a nuestro destino. Así, simplemente cuando se presenta el evento imprevisto, reacomodamos los planes para resolver esta eventualidad. Esta estrategia es muy conveniente ya que es imposible tener un conocimiento completo para cada situación que se nos presenta en la vida diaria.

Puesto que el mundo real es así, contradictorio y cambiante, resulta imperativo que los razonadores aplicados a la solución de problemas reales, sean capaces de lidiar con esas situaciones. Buscando formalizar este tipo de problemas, durante décadas, investigadores de todo el mando se han enfocado en el diseño de nuevos tipos de lógica. Los sistemas de razonamiento basados en la lógica

monotónica son incapaces de ajustar sus conclusiones cuando a su base de conocimiento se agregan hechos contradictorios. En contraste, la característica distintiva de los sistemas de razonamiento basados en lógica no monotónica, es su capacidad para obtener conclusiones válidas y consistentes con la base de conocimiento, ya que ajustan sus conclusiones ante la evidencia de conocimiento contradictorio.

Actualmente, las soluciones semánticas más exitosas, en las cuáles se basan la mayoría de las máquinas de inferencia son la semántica del modelo estable (*Stable Model Semantic*, SME) (Gelfond & Lifschitz, 1988) y la semántica bien fundada (*Well-Founded Semantics*, WFS) (Van Gelder et al., 1991). Ambas semánticas tienen numerosas extensiones.

La Semántica del Modelo Estable se basa en la idea de aceptar múltiples modelos mínimos (*answer sets*) como una descripción del significado de un programa, a diferencia de la lógica clásica, en la cual se acepta un único modelo. La SME tiene sus principios en distintos tipos de lógica como la *default*, la epistémica y la de circunscripción. SME es uno de los métodos más aceptados, ya que permite diseñar programas lógicos con negación por defecto (*Negation as Failure*, NAF or *Negation by Default*). La presencia de NAF da un soporte natural al razonamiento no monotónico, ya que permite expresar excepciones, restricciones y realizar deducciones inteligentes en presencia de conocimiento incompleto.

Esto ha llevado al desarrollo de un novedoso paradigma de la programación, comúnmente referido como *Answer Set Programming* (ASP) (Marek & Truszczynski, 1999). ASP es un paradigma de computación, en el cual las teorías lógicas (cláusulas de Horn con NAF) sirven como especificaciones, y las soluciones de los problemas son representadas por una colección de modelos. Es un lenguaje completamente declarativo y ha sido diseñado también para resolver problemas combinatorios difíciles. La característica más sobresaliente de ASP es que permite representar fácilmente cláusulas por defecto (*default*), con las cuales es posible describir reglas del tipo "normalmente". Esta característica es lo que lo hace tan diferente de otros lenguajes diseñados para la representación del conocimiento. El razonamiento *default* es equivalente al razonamiento de sentido común, y es la forma, en la cual los humanos lidiamos con las incertezas del entorno, tomando decisiones con conocimiento incompleto.

En el diseño de un problema de planificación de tareas para un sistema de manufactura basado en la logica no-monotónica, es posible representar la incerteza epistémica asumiendo que las máquinas

que fueron asignadas a cada trabajo, operarán normalmente durante la ejecución del plan (en este caso incerteza epistémica significa que no hay un conocimiento completo de cómo evolucionará la ejecución del plan). El "normalmente" de la regla se refiere al caso, en el cual las máquinas operan de manera regular y sin interrupciones. Si una máquina se descompone o deja de estar disponible antes de que la tarea termine de procesarse, la entrada del evento externo permitirá que se dispare la regla del plan diseñada para resolver esta situación.

Existen varias máquinas de inferencia para la solución de problemas representados en ASP. La más antigua es *smodels*, pero las más recientes, como Clasp, se benefician de las técnicas que han sido diseñados para resolver los problemas de SAT y CSP (Gebser et al., 2007). Además, dentro de la lógica no-monotónica ASP es una área muy activa y dinámica, donde constantemente se están proponiendo extensiones al lenguaje y mejores técnicas para el desarrollo de resolvedores (Son et al., 2014).

No menos importante resulta destacar el amplio rango de aplicaciones prácticas que han sido resueltas con ASP. Entre las más destacadas se encuentran el sistema para toma de decisiones de una nave espacial (Nogueira et al., 2001), planificacion para generación de equipos en un astillero (Ricca et al., 2012), sistemas filogéneticos (Erdem, 2011).

## 2.5 Aspectos Básicos de ASP

Para explicar la semántica de ASP empezaremos considerando el caso de programas "tradicionales", los cuales, esencialmente son programas Prolog sin variables (Lifschitz, 2008).

### 2.5.1 Sintaxis de ASP

Una regla tradicional es una expresión de la forma:

$$A_0 \Leftarrow A_1 \,,\ldots,\, A_m \,,\, \text{not}\, A_{m+1} \,,\ldots,\, \text{not}\ \ A_n \,, \qquad (2.1)$$

donde $0 \leq m \leq n$ y $A_0 \,,\ldots,\, A_n$ son átomos proposicionales. El átomo es la cabeza de la regla y la lista es su cuerpo. Si el cuerpo está vacío ($n = 0$), entonces a la regla se le llama "un hecho" y es identificado con su cabeza.

### 2.5.2 El "Answer Set" de un Programa Positivo

Un programa es un conjunto finito de reglas tradicionales y reglas que no tienen cuerpo, o reglas con cabeza y cuerpo, pero que no tienen términos negados, por ejemplo:

$p$.

$r \leftarrow p, q$. (2.2)

Una regla tradicional es positiva si $m = n$, ó sea, tiene la forma $A_0 \Leftarrow A_1, \ldots, A_m$ . Un programa tradicional es positivo si todas sus reglas son positivas. Por ejemplo (2.2) es un programa positivo.

La definición de un "*answer set*" de un programa positivo tradicional es: Un conjunto $X$ de átomos satisface un programa positivo tradicional $\pi$ si para todas las reglas sin NAF en $\pi$, $A_0 \in X$para cualquier $A_1, \ldots, A_m \in X$ . Por ejemplo, el conjunto de átomos que satisfacen el programa (2.2) son:

$$\{p\}, \{p, r\}, \{p, q, r\}$$

y su "*answer set*" es $\{p\}$, que es el conjunto más pequeño de átomos que satisface $\pi$.

### 2.5.3 El Answer Set de un Programa con Negación

Para un programa con NAF, un "*answer set*", o modelo estable, se define de la siguiente forma: Teniendo un programa $\pi$ sin variables y un modelo "tentativo" $X$, la definición del reducto de $\pi$ con referencia a $X$ es:

1) Remueve todas las reglas, las cuales contengan átomos con NAF que se contradigan por $X$.

*2)* Remueve todos los átomos con NAF de las reglas remanentes. El programa resultante contiene solamente las reglas de $\pi$ que son aplicables dada $X$. Además, ahora es un programa lógico estándar, sin negación como falla, y el cual admite un modelo único.

3) $X$ es un "*answer set*" si $X$ y $\pi$ coinciden. Generalmente, un programa con NAF admite múltiples conjuntos.

El siguiente es un programa lógico ($\pi$), no tradicional (tiene negación como falla, NAF):

$p \leftarrow \text{not } q.$

$q \leftarrow \text{not } r\ .$ (3)

Dado $\pi$, y si $X = \{q\}$, el reducto de $P$ consiste de un solo hecho, $q$. El “answer set” de $\pi$ es $\{q\}$. Consecuentemente $\{q\}$ es un ”answer set” de $\pi$, y $\{q\}$ es el único “answer set” de $\pi$.

A pesar de su amplia aceptación y extensos fundamentos matemáticos, es hasta recientemente que la semántica del modelo estable se ha popularizado a través de la resolución de problemas prácticos. El éxito reciente se debe a la disponibilidad de motores de inferencia muy eficientes (tales como Clasp, *smodels, Cmodels*) (Maratea et al., 2015), y a los esfuerzos de la comunidad para mejorar y desarrollar nuevas máquinas de inferencia. La evaluación de los mejores motores de inferencia se realiza a través de competencias que se celebran al menos bianualmente. En estas competencias también se evalúan y se premian las mejores prácticas de programación en este campo.

## 2.6 Conclusiones del Capítulo

Aunque aún está lejano el día en que las computadoras puedan superar a los humanos todas las tareas de razonamiento y resolución de problemas. Esto es especialmente cierto en cuanto hace a la generación de conocimiento innovador y a la interacción en mundos dinámicos e inciertos y en actividades altamente creativas. Sin embargo, el desarrollo de teorías y técnicas basadas en lógica para el diseño razonamiento automatizado ha ido creciendo desde 1950, con un notable impulso a partir de los años 80 del siglo anterior, con los avances en lógica no monotónica y el desarrollo de eficientes motores de inferencia basados en SAT. Estos avances han permitido que muchos problemas de utilidad práctica en la industria, la medicina, comunicaciones y muchas otras áreas, puedan ser resueltos gracias a las modernas máquinas de inferencia.

# Capítulo 3

## SISTEMAS DE PRODUCCIÓN EN AMBIENTES DE MANUFACTURA

Este capítulo está dedicado a describir los esquemas de notación para planificación de tareas y actividades basados en los trabajos de Graham (1979) y Brucker (1999). Otro tema importante que se discute es la complejidad computacional, y por último se muestran ejemplos de cómo algunos problemas difíciles de planificación se reducen a otros más simples.

### 3.1 Planificación de Tareas en Ambientes Estáticos: Esquemas de Notación y Clasificación

Durante largo tiempo se han desarrollado de manera paralela, dos tipos diferentes de notación y

esquemas de clasificación para el problema de planificación de tareas. Uno de estos esquemas proviene del enfoque conocido como planificación de máquina *(Machine Scheduling*, MS). El otro, se deriva del problema conocido como planificación de tareas con recursos restringidos (*Resource Constrained Project Scheduling Problem*, RCPSP), el cual tiene un enfoque más general que el de MS. Una característica distintiva de RCPSP es que se concentra en el estudio de problemas, en los cuales se tienen un sólo producto ó bien pequeños lotes de producción, y donde hay que calendarizar recursos escasos para actividades con dependencias temporales. Este tipo de situaciones se da frecuentemente en muchas áreas, y son típicos en los sistemas de manufactura (Hartmann & Briskorn, 2010).

El esquema de clasificación MS se basa en la notación de Graham et al. (1979), denotado por la tripleta $\alpha|\beta|\gamma$, donde el campo $\alpha$ representa los recursos del ambiente, el campo $\beta$ se usa para los procesos, y una medida de eficiencia se denota en el campo $\gamma$.

La simbología para la descripción del ambiente de máquina, campo $\alpha$, se relaciona en la tabla 3.1. Si el número de máquinas es parte de la entrada, le corresponde la simbología identificada bajo el rubro "No. Variable", sino, el que aparece en la columna bajo "No. fijo".

| **Tipo de taller** | **No. Variable** | **No. Fijo** |
|---|---|---|
| 1 sola máquina | 1 | |
| Máquinas paralelas idénticas | *P* | *Pm* |
| Máquinas uniformes | *Q* | *Qm* |
| Máquinas no relacionadas | *R* | *Rm* |
| Máquinas dedicadas | *D* | *Dm* |
| Máquinas multi-propósito | *MPM* | *MPMm* |
| Taller de flujo (*flow shop, conveyor*) | *F* | *Fm* |
| Taller de trabajo (*job shop*). Cada trabajo tiene su ruta determinada | *J* | *Jm* |
| Taller de flujo flexible/híbrido (*flexible/hybrid flow shop*) con *c* etapas. Igual que *F*, pero hay máquinas en paralelo en al menos una etapa y uno de los trabajos requiere procesamiento en una sola máquina (arbitraria) de la etapa | $FF_c$ | |

| | | |
|---|---|---|
| Taller de trabajo flexible (*flexible job shop*) con *c* etapas y algunas máquinas idénticas en cada etapa | *FJc* | |
| Taller abierto (*open-shop*). No hay restricciones de ruta | *O* | *Om* |

Tabla 3.1 Notación de Graham para descripción del taller.

Algunas notaciones para caracterizar los trabajos, campo $\beta$, se detallan en la tabla 3.2.

| **Notación** | **Descripción** |
|---|---|
| *pmtn* | Interrupciones (*preemptions* ) permitidas |
| $r_j$ | Tiempo de liberación (*release times* ) |
| $d_j$ | Fecha límite (*deadline* ) |
| $p_j = 1$ ó $p_j = p$ ó $p_j \in \{1, 2\}$ | Tiempos de procesamiento restringidos |
| *prec* | Restricciones de precedencia arbitrarias |
| *intree - outtree* | Trabajos sin precedencias (*intree*), ó trabajos que no tienen tareas sucesoras (*outtree*) |
| *chains* | Cadenas de precedencia |
| *series-parallel* | Un grafo de precedencia series-paralelo |

Tabla 3.2 Notación de Graham para descripción de los trabajos

Algunas restricciones y limitaciones comúnmente usadas están relacionados con los tiempos de reinicio dependientes de secuencia de los trabajos, como por ejemplo:

- $S_{ijk}$ → tiempo de reinicio entre el trabajo *j* y el trabajo *k* en la máquina *i*;
- $S_{ik}$ → tiempos de reinicio idénticos para todas las máquinas;
- $S_{0j}$ → reinicio para el trabajo *j*;
- $S_{j0}$ → limpieza para el trabajo *j*.

Por cuanto hace a la función objetivo a minimizar, campo $\gamma$, la notación de Graham es detallada en la siguiente tabla 3.3.

| **Descripción** | **Notación** |
|---|---|
| Tiempo máximo de completar los trabajos (makespan) | $C_{\max}$ |
| Máxima latencia | $L_{\max}$ |
| Tiempo total de flujo | $\sum C_j$ |

| | |
|---|---|
| Tiempo total ponderado de flujo | $\sum w_j C_j$ |
| Tiempo total de tardanza | $\sum T_j$ |
| Tiempo total de tardanza ponderada | $\sum w_j T_j$ |
| Número de trabajos tardíos | $\sum U_j$ |
| Número ponderado de actividades tardías | $\sum w_j U_j$ |

Tabla 3.3 Notación de Graham para función objetivo. (Latencia: $L_j = C_j - d_j$; Tardanza: $T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}$, Penalización por unidad: $U_j = 0$ si $C_j \leq d_j$, 1 en otro caso.)

Mientras que la tripleta de Graham ha sido ampliamente reconocida y utilizada para la clasificación de problemas con el enfoque MS desde hace varias décadas, no ocurre lo mismo con RCPSP, ya que durante bastante tiempo no hubo ningún esquema de clasificación que fuese ampliamente aceptado por esa comunidad de investigadores. Los primeros intentos para establecer un nuevo esquema de notación se deben a Herroelen et al. (1997), sin embargo, no es totalmente compatible con el de Graham, y por consiguiente, no se emplea frecuentemente. Esta omisión fue subsanada por Brucker et al. (1999), quienes propusieron un esquema de notación y clasificación unificadas para RCPSP y MS, mismo que ahora se utiliza ampliamente (Pinedo, 2016).

A continuación se resaltará brevemente algunas de las características de este nuevo esquema de clasificación unificado. La tripleta $\alpha|\beta|\gamma$ fue extendida de la siguiente forma: el campo $\alpha$ fue extendido para introducir la notación *PS* (*project scheduling*) ó *MPS* (*multi-mode project scheduling*). Además *PS* se aumenta a *PSm, σ, ρ, (m* recursos, $\sigma$ unidades de cada recurso disponible, cada actividad requiere a lo más $\rho$ unidades de los recursos). En el caso de *MPS* se consideran también recursos renovables y se aumenta su notación a (*MPSm, σ, ρ, μ, τ, ω),* que significa *multi-mode project scheduling* con *m* recursos renovables, $\sigma$ unidades de cada recurso disponible, cada actividad requiere a lo máximo $\rho$ unidades de los recursos, $\mu$ recursos renovables, $\tau$ unidades de cada recurso disponible, y cada actividad requiere a lo más $\omega$ unidades de los recursos. Por lo que respecta al campo $\beta$ de la clasificación de Graham para problemas del tipo *MS*, fue extendida para incluir las siguientes características: tiempos de procesamiento $p_j$, restricciones generales de precedencia (*prec*), tiempos de procesamiento estocásticos ($p_j = sto$), fecha límite para duración del proyecto (*d*), relaciones de precedencia entre las actividades (*chains, intree, outtree, tree*) y restricciones temporales generales dadas por tiempos de retraso mínimos y máximos entre actividades (*temp*). Por último, el campo de la función objetivo $\gamma$, además de las clásicas funciones objetivo como $C_{\max}$, $L_{\max}$, $\sum w_j C_j$ **,** se añadieron funciones para calcular el valor actual neto $\sum C_j^F \beta_j^F$, la nivelación de los recursos $\sum C_j(r_k(S,t))$, y la inversión de los

recursos $\sum C_k \max_k \{r_k(S,t)\}$.

Entre los modelos más comunes de planificación derivados de la notación de Graham, se encuentran: planificación de una sola máquina (*Single Machine Scheduling*, SMS), planificación multiprocesador (*Multiprocessor Scheduling*, MPS), planificación de taller de trabajo (*Job Shop Scheduling*, JSS), planificación de taller abierto (*Open Shop Scheduling*, OSS), planificación de taller de flujo (*Flow Shop Scheduling*, FSS), planificación de taller de flujo flexible/híbrido (*Flexible/Hybrid Flow Shop Scheduling*, FFS/HFS). A continuación mostraremos cómo modelar algunos de estos problemas por un RCPSP.

### 3.1.1 Planificación de Una Sola Máquina

Se tienen *n* trabajos que requieren ser procesados en una sola máquina. El número de trabajos se denota por $j$ =1, ..., $n$. Supongamos que los trabajos tienen precedencia entre sí. Tal problema se representa con RCPSP como $r = 1$, $R_1 = 1$, y $r_{j1} = 1$ para todos los *n* trabajos.

### 3.1.2 Planificación de Máquinas Paralelas

Igual que en el problema anterior, se tienen *n* trabajos y *m* máquinas idénticas $M_1$,....., $M_m$. El tiempo de procesamiento para trabajo $j$, $j$ = 1,…, $n$, es el mismo en cada máquina. Los trabajos deben ser asignados a las máquinas. Este problema corresponde a un RCPSP con $r = 1$, $R_1 = m$, y $r_{j1} = 1$ para todos los *n* trabajos (Fig. 3.1).

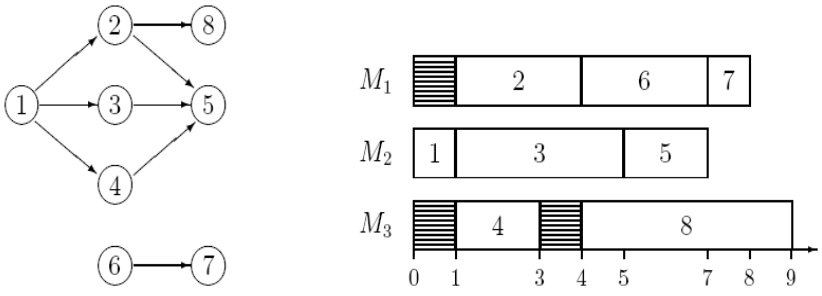



Fig. 3.1. Gráfica de un problema SMS modelado como RCPSP

### 3.1.3 Planificación de Talleres

En problemas generales de este tipo, se tienen $m$ máquinas $M_1,\ldots, M_m$ y $n$ trabajos $j$, $j = 1,...,n$. Cada trabajo $j$ consiste de $n(j)$ operaciones $O_{1j}, O_{2j},\ldots, O_{n(j),j}$ , donde $O_{ij}$ debe ser procesado durante $p_{ij}$ unidades de tiempo en una máquina $\mu \in \{ M_1,\ldots, M_m\}$. Dos operaciones del mismo trabajo no pueden ser procesadas al mismo tiempo.

Para representar el problema general de planificación de un taller como un RCPSP, se consideran $r = n + m$ recursos, $k = 1,..., n + m$, con $R_k = 1$. En tanto que $k = 1, ..., m$ corresponde a las máquinas, se requieren $m+j$ ( $j = 1,..., n$) recursos para modelar las diferentes operaciones del mismo trabajo que no pueden ejecutarse al mismo tiempo. Se requieren también $n(1) + n(2) + ... + n(n)$ actividades $O_{ij}$ donde $O_{ij}$ necesita una unidad del "recurso de máquina" $\mu_{ij}$ y una unidad del "recurso de trabajo" $(m + j)$.

**3.1.4 Taller de trabajo.** Es un problema general de planificación con restricciones de cadena de precedencia de la forma $O_{1j} \rightarrow O_{2j} \rightarrow O_{n(j),j}$ .

**Taller de Flujo.** Es un problema especial de taller de trabajo con $n(j) = m$ operaciones para $j = 1, ..., n$ y $\mu_{ij} = M_i$ , $i = 1, ..., m$, $j = 1, ..., n$ . Existen dos casos de este tipo: a) Si los trabajos tienen que ser procesados en el mismo orden en todas las máquinas, se conoce como **Permutación de taller de flujo.** (Fig. 3. 2); b) El problema de **taller abierto,** el cual es similar al de taller de flujo, pero sin las restricciones de precedencia entre las operaciones (Fig. 3.3).

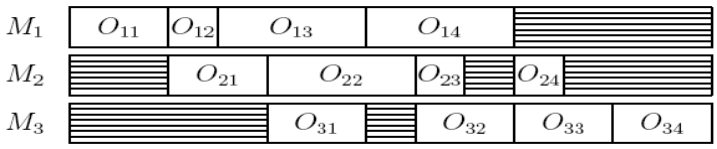



Fig. 3.2. Gráfica de un problema de permutación modelado como RCPSP

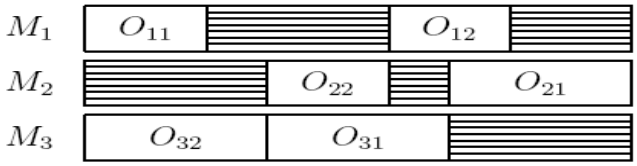


Fig. 3.3. Gráfica de un problema de taller abierto modelado como RCPSP

Algunos ejemplos de la notación de Graham para problemas de planificación, son:

- Fábrica de bolsas de papel → $FF \mid r_j, w_j \mid T_j$ ;
- Tareas en un CPU → $1 \mid r_j, prmp \mid \sum w_j T_j$ ;
- TSP → $1 \mid r_j \mid C_{\max}$ .

### 3.2 Complejidad de Problemas de Planificación de Tareas y Optimización

El diseño de algoritmos eficientes para la solución de problemas de planificación y optimización es una de las tareas más retadoras en el área de computación. Desde que se estableció el área de estudio sobre teoría de la computación, se ha realizado un esfuerzo importante para formular los problemas prácticos como generalizaciones (Johnson, 2012). Algunas de las primeras generalizaciones fueron el problema del vendedor viajero (*Traveling Salesman Problem*, TSP), cómputo de funciones booleanas para el diseño de circuitos, programación lineal y teoría de grafos. Para estos problemas se observó que aunque teóricamente eran fácilmente solubles (puesto que solo se tiene que explorar un espacio de solución finito), en la práctica rápidamente se convierten en intratables porque el tamaño de la solución crece rápidamente, en la mayoría de las veces de una forma exponencial.

Aún y cuando en esa época no se había desarrollado aún el concepto de *complejidad computacional*, ya se había definido que había una posibilidad de "explosión combinatoria", extrema ineficiencia y nula aplicación práctica de los algoritmos de búsqueda de fuerza bruta en la solución, puesto que el número de posibilidades crece inmensamente y requiere de cómputo masivo, a menos que el algoritmo utilice algún tipo de sutileza o conocimiento sobre el problema para reducir la búsqueda (Szeider, 2013).

Otro de los objetivo fundacionales del área de Teoría de la Complejidad, es el de clasificar los problemas de acuerdo a su tratabilidad, es decir, si el problema tiene solución para instancias grandes (Karp, 1986). Sin embargo, existen distintos grados de tratabilidad, los cuales varían de acuerdo al modelo computacional, a los recursos disponibles y a las variantes de las estructuras de datos. Existen muchos problemas que no son tratables, como por ejemplo el problema de las torres de Hanoi, o el problema TSP (Papadimitriou, 1977); en la práctica sólo es posible resolverlos para instancias pequeñas. Aún más, todas las funciones exponenciales y factoriales son no tratables. Como se aprecia en la tabla 3.4, el número de posibles combinaciones para funciones exponenciales y factoriales es de

muchos órdenes de magnitud mayor que el de las funciones polinomiales, por lo que no es suficiente con desarrollar hardware más poderoso, sino que estos problemas deben enfocarse con una mentalidad diferente.

| | ***n*** | | | | |
|---|---|---|---|---|---|
| **Función** | **10** | **50** | **100** | **300** | **1000** |
| $5n$ | 50 | 250 | 500 | 1500 | 5000 |
| $nlog_2 n$ | 33 | 282 | 665 | 2469 | 9966 |
| $n^2$ | 100 | 2500 | 10000 | 90000 | 1000,000 |
| $n^3$ | 1000 | 125,000 | 1,000,000 | 27,000,000 | 1,000,000,000 |
| $2^n$ | 1024 | 16 dígitos | 31 dígitos | 91 dígitos | 302 dígitos |
| $n!$ | 3,628,800 | 65dígitos | 161 dígitos | 623 dígitos | Incontable |
| $n^n$ | 11 dígitos | 85 dígitos | 201 dígitos | 744 dígitos | Incontable |

Tabla 3.4 Funciones y el creciente número de combinaciones de salida para funciones con explosión combinatoria.

Aunque numerosos grupos de investigación estaban dedicados a la búsqueda de algoritmos eficientes para la solución de problemas combinatorios, difícilmente se obtenían soluciones óptimas, no obstante, sí obtenían soluciones adecuadas para muchas aplicaciones prácticas. Por ejemplo, en los sesentas se hicieron las primeras formulaciones de programación dinámica y programación entera para el problema de planificación de tareas (Powell, 2009).

Antes de desarrollar una teoría completa sobre complejidad, la investigación del área de computación estaba principalmente concentrada en la búsqueda de algoritmos con cómputo eficiente. Dada la complejidad inherente a los problemas combinatorios, éstos se convirtieron en un parámetro para diferenciar los algoritmos eficientes de los ineficientes; también fueron cruciales para establecer una noción formal de la complejidad computacional, logro realizado por Hartmanis & Stearns (1965).

Durante un tiempo después del trabajo de Hartmanis & Stearns, siguió creyéndose que era posible encontrar algoritmos eficientes para la solución de problemas con explosión combinatoria. Sin embargo, esta creencia se derrumbó después de que se publicaron los trabajos de Cook (1970) y Karp (1972). En su trabajo seminal, Cook definió formalmente las clases de los lenguajes **P** y **NP**, demostrando que el problema de satisfiabilidad (SAT) es **NP**-completo (**NP**-*complete*). También estableció como un problema abierto ¿**P** = **NP**?

A continuación se mostrará en qué consiste el Teorema de Cook, pero a fin de facilitar la

separación entre clases, se establecerá una asociación entre problemas de decisión y lenguajes formales, puesto que los lenguajes formales son objetos más fáciles de tratar matemáticamente. Para ello es necesaria la siguiente definición:

Sea $\Sigma$ el alfabeto en el que están codificadas las instancias de un determinado problema de decisión $\pi$. El conjunto $\Sigma^*$ está particionado en tres clases: el conjunto $Y_\pi$ de las cadenas que codifican instancias cuya respuesta es "sí"; el conjunto $N_\pi$ de las cadenas que codifican instancias cuya respuesta es "no"; y $R_\pi$ el conjunto de cadenas que no codifican ninguna instancia. Por ejemplo:

Sea $\pi$ el problema que consiste en decidir si un número es primo. Asuma que los datos de entrada se especifican en binario; entonces el lenguaje asociado es:

$Y_\pi = \{10, 11, 101, 111, \ldots\}$.

Con esto vemos que es factible una correspondencia entre los problemas de decisión y los lenguajes formales.

### 3.2.1 Teorema de Cook:

- Probar que SAT $\in$ NP, mediante el siguiente teorema

  Un lenguaje $L \in$ NP si y solo si existe un lenguaje $L_{ck} \in$ P tal que

  $L = \{x: \exists_y (x,y) \in L_{ck}$ , además $| y | \leq p (| x |)\}$, donde $p$ es un polinomio.

- Luego se toma un lenguaje arbitrario $L \in$ NP, y se muestra que $L \leq_p$ SAT, donde $L \leq_p$ SAT es una función de reducibilidad que permite definir una comparación entre lenguajes. Es decir, que es posible reducir el lenguaje $L$ a SAT y además ésta es una relación de pre-orden.

- Se define primero la reducción: sea $T = \langle Q, \Sigma, \Delta, q_0, q_1 \rangle$, la Máquina de Turing No Determinística (MTND) que acepta a $L$, y $x \in \Sigma^*$. Como $L \in$ **NP,** sabemos que $T(x)$ termina (aceptando o rechazando) en tiempo polinomial $p(|x|)$.

- A continuación se define una fórmula bien formada (fbf), $F(T, x)$, tal que $F(T, x)$ sea satisfacible si y solo sí $T(x)$ termina en el estado aceptador.

### 3.2.2 Implicaciones del Teorema de Cook:

- Si se encuentra un algoritmo polinomial para SAT, usando la poli-transformación de Cook, es posible resolver todos los problemas de la clase **NP** en tiempo polinomial, **y consecuentemente se probaría también que P=NP**.
- Si se encuentra una poli-transformación desde SAT a otro problema *Z*, entonces *Z* es también un problema **NP**-completo, o viceversa. Si se encuentra un algoritmo polinomial para *Z*, usando una poli-transformación de SAT a *Z*, sería posible resolver cualquier instancia de SAT en tiempo polinomial y por lo tanto también sería posible resolver todos los problemas de la clase **NP**.

Desde que Cook planteó su famoso teorema, se ha buscado la respuesta al problema, ¿es **P** = **NP***?* Esta búsqueda ha sido parte central de la investigación del área de la teoría de complejidad (Cook, 1971). Casi inmediatamente después del trabajo seminal de Cook, Karp (1972) hizo otra importante aportación, mostrando que las versiones de decisión de muchos problemas de optimización muy conocidos, son **NP**-completos, y que la única forma de encontrar algoritmos eficientes para resolver estos problemas es probar si es cierto que **P**=**NP**, lo cual hasta la fecha es un problema abierto (Fortnow, 2009), (Wigderson, 2010)..

Estas aportaciones tuvieron un efecto dual. Por un lado, un grupo de investigadores (Papadimitriou, 1997), (Hartmanis, 2001) se enfocaron en comprender la estructura de la clase **NP** y al desarrollo de elegantes teorías, tales como la conjetura de Berman-Hartmanis que establece que todos los conjuntos **NP**-completos, son isomorfos y por lo tanto tienen una estructura similar. Un segundo grupo de investigadores (Johnson, 1974; 2002; Blum et al., 2011) dirigieron sus esfuerzos a la búsqueda de algoritmos ya no eficientes, sino aproximados pero lo suficientemente veloces para que resultasen adecuados para aplicaciones prácticas.

A continuación resumiremos brevemente las características de las clases de complejidad más importantes. Para ilustrar estas ideas se acompaña también en la figura 3.1 un esquématico de estas clases.

### 3.2.3 Algoritmos Polinomiales

Un problema es llamado soluble polinomialmente si se resuelve por algoritmos polinomiales. Por ejemplo, $1\|\sum w_j C_j$ se resuelve planificando los trabajos de acuerdo a un ordenamiento de valores $w_j$

/$p_j$ no-crecientes.

### 3.2.4 Las Clases P y NP

En un problema de decisión, la salida es *Falso/Verdadero*. Un problema de optimización puede ser resuelto como si fuera un problema de decisión. También es posible asociar un problema de planificación con un problema de decisión definiendo un umbral $k$ para la función objetivo $f$. El problema de decisión consiste en determinar si existe un plan conveniente $S$ que satisfaga $f(S) \leq k$.

Los problemas solubles por un algoritmo determinístico en tiempo polinomial pertenecen a la clase **P**. Los problemas de la clase **NP** son aquellos que son solubles por un algoritmo no determinístico en tiempo no-polinomial. Ahora bien, como es imposible desarrollar algoritmos no-determinísticos, una manera equivalente de verificar si un problema dado pertenece a la clase **NP** es la siguiente: cuando se tiene una propuesta de una solución, esta debe ser verificada por un algoritmo en tiempo polinomial. La solución propuesta es llamada un *certificado* para la entrada de la instancia del problema. Las versiones de decisión del problema de planificación pertenecen a la clase **NP**, puesto que una respuesta "sí" es *certificada* por un plan factible $S$ con $f(S) \leq k$.

Por ejemplo, el problema del circuito Hamiltoniano (*Hamiltonian Circuit*, HC) (Garey et al., 1976) consiste en encontrar un ciclo en un circuito que conecte todos los nodos, pero con la condición de que ningún nodo sea repetido en el ciclo. Un algoritmo debería recorrer la trayectoria certificada y verificar si el primer nodo coincide con el último, y si no han sido visitado dos veces o más los mismos nodos. Después verificaría si todos los nodos del grafo forman parte de la trayectoria, y si todos los arcos en la ruta conforman el grafo de entrada. Esto toma tiempo polinomial con respecto al número de nodos $N$, y el número de vértices $V$, del grafo que representa el circuito. Por esta razón, el problema HC está en la clase NP.

### 3.2.5 Problemas NP-completos y Problemas NP-duros

Para dos problemas de decisión $P$ y $Q$, decimos que $P$ se reduce a $Q$ (denotado por $P \propto Q$ ) si existe una función $g$ computable en tiempo polinomial que transforma las entradas de $P$ en entradas para $Q$, bajo el siguiente criterio:

$x$ es una entrada "sí" para $P \leftrightarrow g(x)$ es una entrada "sí" para $Q$.

Las reducciones polinomiales tienen las siguientes propiedades:

1) si $P \alpha Q \rightarrow Q \in P$ implica que $P \in \mathbf{P} \equiv P \notin \mathbf{P}$ implica que $Q \notin \mathbf{P}$;

2) si $P \propto Q \ \& Q \propto R \rightarrow P \propto R$.

### 3.2.6Propiedades de los problemas NP-duros

- Si existe una cadena de poli-transformaciones entre dos problemas de decisión o entre SAT y un problema *Z*, o viceversa, estos problemas son **NP**-duros (**NP**-*hard*). SAT fue el primer problema en ser identificado como **NP**-duro.
- Si un problema **NP**-duro también pertenece a la clase **NP**, es llamado **NP**-completo. El conjunto de estos problemas es llamado **NP-**completo.

## 3.3 Algunos Problemas de Planificación Determinísticos

Actualmente, para sobrevivir a sus competidores, una empresa tiene que enfrentar y resolver diversos retos. Uno de los cruciales es precisamente el de cumplir en tiempo y forma con los productos demandados por sus clientes. Es por ello que los sistemas de producción deben ser cuidadosamente planificados, de manera tal que se haga uso eficiente de los recursos que se utilizan en la fabricación de los productos.

Una manera de recortar los gastos de producción es la de tener un plan optimizado. Pero la mayoría de los problemas de optimización pertenece a la clase **NP-**duros, y por ende, éstos son intratables. Por esta razón, siempre se trata de reducir la dificultad del problema, convirtiéndolo a otro problema que sí sea tratable.

Sin embargo, algunos problemas son casos especiales de otros problemas. Para mostrar algunas reducciones posibles, primero establecemos la siguiente notación:

$\alpha_1 \mid \beta_1 \mid \gamma_1 \quad \propto \quad \alpha_2 \mid \beta_2 \mid \gamma_2$

Por ejemplo:

$1 \| \sum C_j \ \propto 1 \| \sum w_j C_j \ \propto \ Pm \| \sum w_j C_j \ \propto \ Qm \mid prec \mid \sum w_j C_j.$

Algunos casos de reducción compleja, son los siguientes:

$\alpha_1 \mid \beta_1 \mid \gamma_1 \quad \propto \quad \alpha_2 \mid \beta_2 \mid U_j$ ,

$\alpha_1 \mid \beta_1 \mid \gamma_1 \quad \propto \quad \alpha_2 \mid \beta_2 \mid T_j$ .

A menudo los problemas solubles polinomialmente, se resuelven de manera eficiente por algoritmos polinomiales. En cambio, para los problemas **NP**-dificiles, es muy improbable que existan algoritmos polinomiales que permitan su resolución. Para lidiar con la complejidad de los problemas **NP**-dificiles, usualmente se utilizan reducciones elementales. Por ejemplo, el problema descrito en primer término en la siguiente ecuación, se reduce al siguiente problema descrito allí mismo:

$1|\ tree,\ p_i=1|\sum C_j \ \propto\ 1|\ prec,\ r_i,\ pmtn|\ \sum w_j C_j$ .

Esta reducción es posible por lo siguiente:

a) los arboles de precedencia (*tree*), son un caso especial de las precedencias generales;

b) si $r_j = 0$ y $p_j = 1$ , para todos los trabajos, entonces un plan interrumpible (*preemptive*) se transforma en uno no-interrumpible sin incrementar el objetivo de la función;

c) $\sum C_j$ es un caso especial de $\sum w_j C_j$.

A continuación se muestran los grafos de las reducciones elementales para problemas de una sola máquina, respecto del ambiente de máquina, las funciones objetivo y los trabajos.

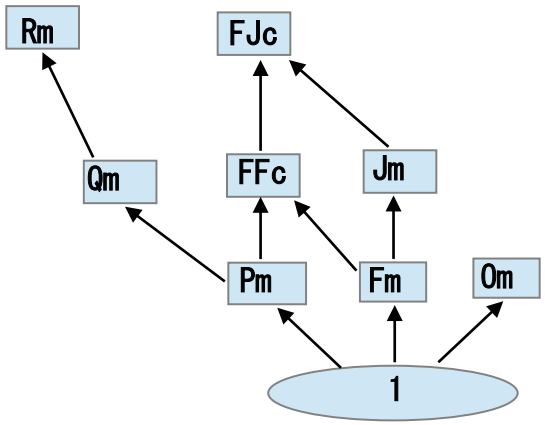



Fig. 3.4 Reducciones para ambiente de máquina

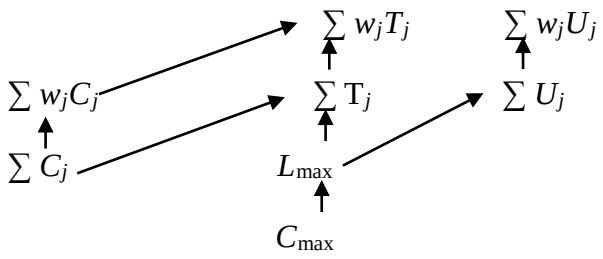



Fig. 3.5 Reducciones para funciones objetivo

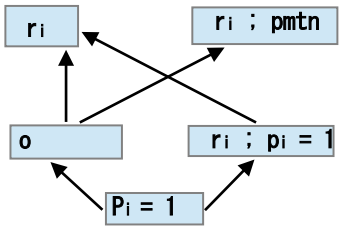



Fig. 3.6 Reducciones para trabajos

### 3.4 Conclusiones del capítulo

Aunque la solución óptima de problemas de planificación y optimización continúa como un problema sin solución, es importante seguir investigando sobre este tópico. El uso de máquinas de inferencia para solución general de problemas combinatorios como SAT, luce prometedora a la luz de las nuevas líneas de investigación. Por ejemplo, se ha descubierto que las instancias de problemas combinatorios derivadas de aplicaciones reales, muestran ciertas propiedades estructurales que son utilizadas para diseñar algoritmos que resuelvan el problema en tiempo polinomial. Algunas de estas propiedades estructurales son estáticas, mientras otras son dinámicas. Una de estas propiedades estáticas es el grado de aciclicidad en los grafos de restricciones. Por ejemplo, un problema combinatorio es soluble en tiempo polinomial si el ancho del árbol de su grafo de restricciones tiene límites constantes (Dilkina, Gomes & Sabharwal, 2007), (Gomes et al., 2000). En cambio, el enfoque basado en propiedades estructurales dinámicas se centra en buscar estructuras escondidas a las que se les conoce como conjunto de puertas traseras (*backdoor set).* La investigación sobre este tópico ha demostrado hechos sorprendentes, ya que se ha descubierto que muchos instancias de problemas reales tienen conjuntos pequeños de variables que conforman puertas traseras y por ello, una vez que un motor de

razonamiento SAT instancía estas variables, el resto se resuelva fácilmente. Es decir la localización e instanciación de estos conjuntos transforma un problema **NP**-completo a uno tratable (Samer & Szeider, 2010). Si bien, la mayoría de estas teorías y técnicas fueron desarrolladas inicialmente para las máquinas de inferencia SAT, después éstas han sido aplicadas en el diseño de otros motores de razonamiento de diferentes lenguajes formales, tales como CP y ASP, e inclusive han desarrollado sus propias técnicas con el mismo propósito, explotando las características propias de estos lenguajes (Fichter & Szeider, 2015), (Kronegger et al., 2014), (Gaspers et al., 2014). En consecuencia, los razonadores de CP y ASP también resuelven en tiempo polinomial muchas instancias provenientes de problemas reales que de otro modo son intratables.

# Capítulo 4

## ESTADO DEL ARTE SOBRE PLANIFICACIÓN/REPLANIFICACIÓN

En este capítulo se presenta un análisis del estado del arte sobre enfoques en solución de problemas de replanificación en sistemas de manufactura, especialmente en los sistemas de manufactura de semiconductores. Previamente a la discusión se plantean cuestiones importantes acerca de la conveniencia de abordar el modelado y solución del problema de planificación con un enfoque robusto en lugar de uno óptimo. Los resultados del estudio fueron clasificados de acuerdo a las taxonomías propuestas por los autores de los artículos relacionados.

### 4.1 Planificación Óptima versus Replanificación Robusta

Tradicionalmente la investigación en el área de la planificación ha estado enfocada principalmente en dos objetivos: optimalidad y eficiencia. El uso óptimo de tiempo y recursos tiene un beneficio directo en la reducción de costos de producción, mientras que la eficiencia está relacionada a los recursos computacionales. Esto implica que cada día sean necesarios algoritmos más veloces y eficientes. Sin embargo, no es práctico dedicar demasiado esfuerzo en buscar como único objetivo la optimalidad de la planificación, puesto que ésta raramente se ejecuta sin perturbaciones. Además, los resultados de varias investigaciones arrojan que en términos de costos, un plan robusto es más importante en lugar de uno óptimo. En otras palabras, si el principal objetivo es mantener la estabilidad del sistema, entonces el objetivo más importante es asegurarse que el nuevo plan requiera un mínimo en la relocalización de recursos.

Otro concepto relacionado a la robustez y estabilidad es el "nerviosismo" en el taller de piso (*nervousness*). Un plan estable es aquel que tiene desviaciones mínimas respecto al original. Generalmente, la desviación se mide como la diferencia entre los tiempos de terminación planificados inicialmente y los tiempos reales de terminación (Gabrel et al., 2014), (Sabuncuoglu & Goren, 2009). Por contraposición, el nerviosismo es el opuesto a la estabilidad. Una de las formas de medirlo es contando el número de relocalizaciones que se requieren para ajustar el nuevo plan con respecto al anterior. Resultados experimentales muestran que el "nerviosismo" derivado de una replanificación es una situación indeseable, porque los costos de producción se incrementan e influyen negativamente en el tiempo mínimo de terminación de los trabajos (*makespan*) (Verfaillie, 2005).

Recalcamos de nuevo que la investigación sobre la planificación está enfocada en optimalidad y eficiencia mientras que replanificación lo está en robustez. Bajo el punto de vista clásico de planificación, solo los ambientes estáticos buscan la optimalidad, puesto que el plan solo es óptimo si durante su ejecución no existe ninguna interrupción que lo torne inviable. Para ambientes cuasi-determinísticos, es ventajoso asumir que el ambiente es estático, puesto que problemas de esta clase son más tratables (Sabuncuoglu & Goren, 2009). En contraste, hay evidencias que demuestran que para ambientes altamente inciertos, es más conveniente el diseño de planes robustos en vez de optimizados (Fitkov, 2010).

En la literatura actual, se identifican dos enfoques diferentes en el modelado de problemas no

determinísticos: 1) El investigador usa tanta información acerca de incertezas como disponga. Programación Estocástica (Bonfill et al., 2008), (Ghezail et al., 2010), y Optimización Robusta (Mondal et al., 2013) pertenecen a este enfoque. 2) El problema es resuelto dividiéndolo en dos partes: una capa del software lidia con la naturaleza dinámica del problema, mientras que la otra resuelva la parte determinística. Este enfoque es llamado Planificación Dinámica (*Dynamic Scheduling*) ó Planificación Reactiva (*Reactive Scheduling*) (Larsen & Pranzo, 2012).

Ambos enfoques tienen pros y contras. Programación Estocástica y Optimización Robusta han reportado mejores resultados que el segundo enfoque. Esto se debe a que cuando se incorporan incertezas al modelo, éste es más cercano a la realidad. Desafortunadamente, la incorporación de incerteza al modelo es costosa ya que el problema resultante pertenece a una clase de complejidad más alta. Mientras la complejidad de un problema de la planificación sin incertezas pertenece a la clase de problemas **NP**-completos, uno estocástico pertenece a la clase de problemas **PSPACE**-completos (Papadimitriou, 1985).

Las principales ventajas del enfoque dinámico son: 1) Tratabilidad, la cual se logra partiendo el problema en dos entidades diferentes, la parte del planificador (*scheduler*) y la parte reactiva. De este modo es posible resolver eficientemente algunas instancias de un problema mediante el uso de un motor de inferencias determinístico; 2) Se obtienen soluciones más veloces con los motores de inferencias disponibles; 3) Simplificación del modelado de la incerteza, ya que no es necesario conocer las distribuciones de probabilidad de las variables – información que en general es muy difícil de obtener.

Las desventajas del enfoque dinámico son: a) la replanificación dinámica no incorpora las incertezas durante el modelado del problema, y consecuentemente, el plan resultante es menos robusto que el obtenido usando el primer enfoque; b) Hay pocos resultados teóricos reportados a la fecha. Algunos motores de inferencias recientes para replanificación reactiva fueron propuestas en (Gebser et al., 2012), (Larsen & Pranzo, 2012).

### 4.2 Casos de Estudio de Replanificación

Para realizar el modelado de problemas de planificación bajo incerteza, primeramente hay que fijar las estrategias y políticas de replanificación. Se han propuesto diferentes esquemas para clasificación de estrategias y políticas de replanificación (Vieira & Lin, 2003), (Aytug et al, 2005), (Ouelhadj &

Petrovic, 2009), pero hasta el día de hoy no existe un consenso general para definir el mejor esquema. Las taxonomías de replanificación de Vieira (2003) y Aytug (2005) tienen características en común; Ouelhadj y Petrovic (2009) añadieron una nueva estrategia de planificación, haciendo un total de cuatro estrategias: completamente reactiva, predictiva-reactiva, robusta predictiva-reactiva y robusta pro-activa. Consideramos que la taxonomía de Ouelhadj (2009) es más precisa porque hace una separación más fina entre las estrategias. Por esa razón, hemos decidido usarla para clasificar los sistemas estudiados en esta revisión de la literatura.

En los problemas de planificación con una estrategia total reactiva, la incerteza no es tomada en cuenta al momento de realizar un plan parcial. Así, si ocurre un evento inesperado, el plan se re-evalúa o re-optimiza. En las otras tres estrategias de replanificación, se embeben diferentes niveles de incerteza para conformar un plan predictivo, y las interrupciones causadas por eventos inesperados se atienden de acuerdo a las políticas de replanificación establecidas previamente. Además, tanto en el caso de estrategias del tipo robusto predictivo-reactivo como en las de tipo robusto pro-activo, los objetivos adicionales que se toman en cuenta son la robustez y la estabilidad.

En las siguientes secciones, se estudian algunos casos de replanificación y se reportan los resultados clasificados de acuerdo a la estrategia que se usó para lidiar con las interrupciones.

### 4.2.1 Planificación Reactiva

En una planificación completamente reactiva, cada vez que se atiende una interrupción, se ajusta el plan siguiendo las reglas de despacho o las heurísticas establecidas de antemano. Las metodologías de modelado para planificación reactiva de SMS son más sofisticadas que las de los sistemas de manufactura más simples. Es por ello que, difícilmente los investigadores intentan construir un modelo completo de una SMS usando métodos matemáticos ortodoxos (Mönch et al., 2011). En su lugar, los investigadores usan frecuentemente técnicas de modelado tales como lenguajes que permitan describir sistemas dinámicos. Un lenguaje popular para el modelado de sistemas complejos es Redes de Petri. Otros enfoques reportados en la literatura es la Teoría de Colas, Teoría Browniana, Cadenas de Markov, Teoría de Kelly, y los modelos de Flujo Continuo (Siddique & Adeli, 2016), (Wen et al., 2001), (Sohl & Kumar, 1995), (Kumar, 1994), (Schrijver, 2012).

Para obtener la información acerca del estado actual de una fábrica, es necesario modelar el comportamiento del sistema, incluyendo una de las características más relevantes de SMS, la re-entrancia (el patrón de flujo de sus líneas de manufactura). Por ejemplo, un problema de planificación para una fábrica de obleas fue modelado y simulado mediante una combinación de teoría de colas y redes de Petri coloreadas-temporizadas (*Color-Timed Petri Nets*, CTPN) (Wen et al., 2001). El planificador fue implementado por un algoritmo genético (*Genetic Algorithm*, GA) que dinámicamente busca la solución por una regla de despacho apropiada. Los resultados experimentales mostraron que el planificador basado en GA tiene una eficiencia superior comparado con las reglas de despacho tradicionales. Otros autores (Qiao et al., 2013), reportan un caso de planificación de un SMS mediante redes de Petri jerárquicas coloreadas-temporizadas (*Hierarchical Color-Timed Petri Nets*, HCTPN) y GAs extendidos (*Extended Genetic Algorithms*, EGA). El objetivo de esta investigación se centró en estudiar, cómo optimizar la combinación de políticas de planificación. La simulación del sistema mostró resultados casi óptimos.

De manera similar, en (Lee et al, 2009), (Liu et al., 2009), se usó un enfoque basado en redes de Petri temporizadas y extendidas bajo el paradigma orientado a objeto (*Time Extended Object-Oriented Petri nets*, TEOPN) para modelar la simulación de un Sistema de Fabricación de Obleas de Semiconductor (*Semiconductor Wafer Fabrication System, SWFS*). Las TEOPNs fueron usadas para describir la SWFS como una serie de objetos. Coincidentemente, en ambos resultados reportados por los equipos de investigadores arriba indicados, las reglas de despacho fueron implementadas mediante un algoritmo dinámico para cuello de botella. Liu et al. evaluaron la eficiencia usando una arquitectura de simulación SWFS. Mientras Lee et al. propusieron un nuevo enfoque para planificación multi-objetiva con despacho en tiempo real *(Multiple-Objective Scheduling and Real-Time Dispatching*, MSRD), el cual básicamente consiste de dos módulos principales: el primer módulo es un planificador multi-objetivo fuera de línea y el segundo módulo, es un despachador en línea encargado de atender las interrupciones en tiempo real. La evaluación de la eficiencia fue hecha vía una simulación construida sobre una plataforma conformada por el prototipo MSRD y las TEOPNs. La SWFS virtual fue derivada de una fábrica real localizada en Shangai. En ambos estudios se obtuvieron resultados similares que muestran que la planificación con despacho dinámico tiene una mejor eficiencia, comparativamente, respecto de las siguientes políticas de despacho: a) porcentaje crítico y FIFO (*Critical Ratio*, CR + *First In First Out*, FIFO), y b) fecha de entrega más temprana (*Earliest Due Date*, EDD).

Una aproximación al modelado del problema de re-entrancia en la manufactura de

semiconductores es reportada por Coron et al. (2010). Estos autores caracterizaron el problema de re-entrancia como un problema de control óptimo gobernado por la ley escalar de conservación hiperbólica, usando ecuaciones diferenciales parciales para su solución.

Otra metodología que parece bien situada para resolver problemas de planificación complejos, tales como los de SMS, son los sistemas multi-agentes, (*Multi-Agent Systems*, MAS). Muchos investigadores han tomado ventaja de las capacidades superiores de MAS para lidiar con la aleatoriedad y el dinamismo de problemas complejos para realizar la planificación en ambientes de manufactura (Framiñan & Ruiz, 2010), (Vrba et al., 2012) y (Zhang & Anosike, 2012).

Lin and Long (2011), desarrollaron y probaron una plataforma de simulación distribuida para una parte del proceso de producción y utilizaron datos reales de una fábrica de semiconductores basada en Shangai. La arquitectura de la plataforma fue estructurada en tres capas: la capa de red de comunicación, la capa intermedia entre el hardware y el software (basada en JADE), y la capa de simulación multi-agente. En tanto, Mönch et al. (2011) propusieron una nueva arquitectura de un sistema basado en agentes para el control de producción de un SMS, el cual fue implementado como un esquema jerárquico multi-capa.

A fin de cumplir con los abrumadores requisitos de los modernos sistemas de producción y tomando en cuenta que a la vez las empresas permanezcan competitivas, la planificación y el control de su ejecución deben permanecer firmemente acoplados. Una fuerte integración en los sistemas de ejecución de manufactura (*Manufacturing Execution Systems*, @MES) es esencial para conseguir un comportamiento adaptivo debido a las interacciones entre un conjunto de agentes actuando como administradores autónomos, como ocurre en las herramientas de modelado y simulación basado en agentes (*Agent Based Modeling and Simulation*, ABMS), propuestas por Rolón et al. (2012) para el diseño de un @MES distribuído. Rolón usó un enfoque basado en una técnica bio-inspirada denominada sistemas holónicos de manufactura (*Holonic Manufacturing Systems*, HMS) (Giret & Botti, 2004), (Valckenaers et el., 1994). Los agentes mostraron comportamientos emergentes comparables a los de un sistema complejo adaptivo.

Además, los agentes están bien situados para procesos de modelado, donde cada agente debe adaptar y modificar su propio comportamiento sobre el tiempo. Cada agente tiene autonomía para resolver las disrupciones relacionadas a su función. El cumplimiento de metas es logrado mediante trabajo colaborativo. Los agentes se comunican entre sí por medio de una gráfica de Gantt dinámica.

De acuerdo a los resultados de simulación, los mecanismos de interacción entre los agentes fueron estables y robustos a pesar de la total autonomía de todos los agentes y la ausencia de un plan maestro.

### 4.2.2 Planificación Predictiva-Reactiva

La estrategia predictiva-reactiva ha sido aproximada de varias maneras por los investigadores, pero generalmente los planes iniciales son diseñados fuera de línea. Si al tiempo de ejecución ocurre un evento no anticipado, se realiza una replanificación parcial o total basada en las políticas establecidas previamente.

Hung et al. (2013), usaron la estrategia predictiva-reactiva para un problema de planificación en el área de fotolitografía de una fábrica de obleas para semiconductores. Su meta fue comparar la efectividad y la eficiencia de tres algoritmos en la implementación de una replanificacion óptima. Los algoritmos son: recocido simulado (*simulated annealing*), los GAs y la búsqueda tabú. Ellos también propusieron un método de búsqueda de sensitividad para mejorar la eficiencia, que consistía en: independientemente del momento cuando se requiera reparar el plan, usar como punto de inicio el plan inicial para la búsqueda de un nuevo plan. Los resultados experimentales mostraron que la búsqueda sensitiva mejoró la eficiencia. El algoritmo que mostró mejor desempeño en este experimento fue búsqueda tabú.

Recientemente, nuevas teorías y metodologías están siendo introducidas para la representación del conocimiento sobre el dominio de problemas de replanificación. Muñoz et al. (2011), reportaron un caso de estudio de una fábrica de químicos con lotes multi-producto. El evento inesperado que se tomó en consideración era el incremento en el tiempo de procesamiento. La función objetivo considerada es la maximización de la ganancia de la planta. El modelado del problema estuvo basado en modelos dinámicos aproximados y ontologías. Usualmente, la información en línea y la histórica son independientes entre los diferentes niveles de decisión y no son propiamente integradas durante el proceso de replanificación. En cambio, el autor utilizó ontologías para integrar este tipo de información.

Las crecientes necesidades del mercado han obligado a repensar sobre los procesos de producción. Para copar con estas necesidades, ha surgido una tendencia de desarrollar líneas de producción mixtas. Bajo este esquema de producción, la orden del cliente consiste de productos muy variados pero con volumen bajo. Típicos sistemas de manufactura con producción de línea mixta son

las industrias de dispositivos electrónicos y comunicación inalámbrica. Huang et al. (2013), resolvieron un problema de planificación para esta clase de sistemas de manufactura. Para programar la producción de la planta y maximizar el margen operativo, los autores adoptaron una técnica basada en teoría de restricciones (*Theory of Constraints*, TOC). Esta técnica es más conocida por su designación en inglés (*Drum-Buffer-Rope*, DBR) y permite gestionar de forma global todas las operaciones. La aplicación de estas técnicas ayuda a los directivos a detectar tempranamente problemas de producción, y evaluar la conveniencia de realizar la replanificación por adelantado. Otro objetivo fue comparar las técnicas DBR y EDD, y estudiar el impacto en la eficiencia de las órdenes retrasadas, la máxima tardanza, los costos y la totalidad del flujo hasta su finalización. Los resultados mostraron que DBR fue mejor que EDD porque minimizó la tardanza más larga en las órdenes de los clientes y ofreció una mayor flexibilidad y habilidad para administrar una fábrica con sobrecarga de capacidad y frecuentes disrupciones.

### 4.2.3 Planificación Robusta Predictiva-Reactiva

La planificación de sistemas altamente estocásticos es problemática porque se presentan comportamientos impredecibles. Sin embargo, la re-planificación ante cada disrupción tampoco es conveniente. En vez de aplicar una política de replanificación, cada vez que ocurre una disrupción, una mejor opción es replanificar justo cuando la dimensión de la interrupción torna no viable el plan en ejecución (Church & Uzsoy, 1992). La solución óptima y genérica para un problema de este tipo es aún una cuestión abierta. Por ello, múltiples enfoques y métodos constantemente están siendo propuestos y probados.

En una investigación, autoría de Vonder et al. (2007), se reportaron los resultados de un experimento diseñado para la evaluación de planes sobre proyectos predictivos reactivos de recursos restringidos *(Resource-Constrained Project Schedule Problem*, RCPSP). La función objetivo es la minimización del *makespan*. Otro objetivo es garantizar que la robustez del plan no sea afectada por los eventos aleatorios. Básicamente, este problema de planificación es una versión estocástica del estático, puesto que la duración de las actividades no tiene duración fija. El experimento completo de Vonder consistió en evaluar todas las posibles combinaciones de tres métodos de planificación básicos, en conjunto con cuatro procedimientos de replanificación reactivos. El impacto en la eficiencia es medido de acuerdo a las variaciones de rango de los siguientes parámetros: 1) duración; 2) ponderación ó la pseudo-actividad final (*dummy*) con respecto al promedio del resto de las actividades; y, 3)

probabilidad de terminación del proyecto en la fecha límite (*Timely Project Completion Probability*, TPCP).

Los métodos básicos de planificación que fueron evaluados son: 1) RCPSP-predictivo: un procedimiento exacto con la duración promedio de las actividades; 2) un procedimiento subóptimo de heurísticas simples de planificación basadas en prioridad, específicamente en una regla que prioriza el tiempo más tarde de inicio (*Latest Start Time, LST*); 3) flujo de recursos con factor flotante-dependiente (*Resource Flow Dependent Float Factor*, RFDFF) que incluye procedimientos subóptimos dirigidos a minimizar la estabilidad de la función de costo. Los cuatro métodos reactivos de planificación son: 1) RCPSP-reactivo: se hace una replanificación completa mediante un procedimiento exacto, donde solamente las actividades terminadas al momento de la disrupción son tomadas en consideración; Flujo Fijo (*Fix Flow*): se realiza una replanificación parcial bajo el concepto del termino denominado en inglés como *railway*, o sea, que las actividades nunca empiezan más temprano que el punto de inicio asignado en el plan base; 3) reglas de prioridad basadas en actividad (*Activity-Based PriorityRrules*, ABR): el problema se resuelve por heurísticas, y la solución es más una lista de actividades, que un plan; 4) el problema de planificación de un proyecto con recursos restringidos anticipados-tarde (*Resource-Constrained Earliness-Tardiness Project Scheduling Problem*, RCPSPWET): usando procedimientos exactos, se minimizan los costos de trabajos anticipados-tarde.

La conclusión final de este experimento fue que aún y cuando se usaron procedimientos exactos para generar planes proactivos y reactivos, el objetivo de optimización del TPCP mostró mejores resultados. Sin embargo, el objetivo de estabilidad no fue mejorado. En general, considerando los resultados obtenidos en todos los experimentos, los autores llegaron a la conclusión que cuando se tienen requisitos muy justos, fechas de entrega no tan ajustadas, y valores bajos respecto a la variabilidad de la duración, una mejor estrategia es generar una planificación robusta proactiva basada en las heurísticas RFDFF. Sin embargo, para ambientes altamente variables y TPCP con valores bajos, las heurísticas RFDFF no son la mejor opción. En estos casos, es preferible combinar un procedimiento que genera un plan base de duración mínima en conjunto con una política reactiva como WET para mejorar la estabilidad. En conclusión, los autores aconsejan conducir más estudios acerca de planificación robusta proactiva a fin de mejorar los resultados de WET.

Kuster et al. (2010), propusieron un enfoque genérico para planificación parcial en ambientes realistas altamente estocásticos. Primero, ellos propusieron una extensión del marco conceptual RCPSP

(x-RCPSP) para describir formalmente los problemas derivados de administración de disturbios (*Disturbance Management Problems*, DMP). En esencia, el x-RCPSP conceptualizó elementos *activos* y elementos *inactivos*. Solamente los elementos activos fueron considerados para la replanificación. Los autores también propusieron una replanificación local (*Local Rescheduling*, LRS) para construir la planificación parcial. LRS está basado en ventanas de tiempo que se extienden de forma bidireccional para hacer la búsqueda de soluciones potenciales. Estas soluciones potenciales deben cumplir con los nuevos requisitos impuestos por la ocurrencia de eventos estocásticos.

Un enfoque y objetivos similares fueron propuestos por Huang et al. (2010), en su investigación sobre reparación de planificación para un taller de trabajo (*Job Shop Scheduling Repair*, JSSR). La meta de este problema fue obtener una reparación de la planificación estable mediante un compromiso entre la optimización del *makespan* y la desviación de la eficiencia durante la replanificación. El problema fue formulado con disyunción temporal (*Disjunctive Temporal Problem*, DTP), y conformado como un problema de satisfacción óptima de restricciones (*Optimal Constraint Satisfaction Problem*, OCSP). La solución del problema se realizó mediante un algoritmo que integra consistencia incremental y generación eficiente de candidato.

### 4.2.4 Planificación Robusta Pro-Activa

En esta estrategia, el objetivo es obtener un plan robusto y estable. Para conseguir esta meta, los planes robustos y estables son creados incluyendo las incertezas fuera de línea. En teoría, la variante resultante debería ser insensible a las disrupciones. Sin embargo, inevitablemente ocurren algunos eventos exógenos no anticipados durante la ejecución del plan proactivo. Frecuentemente, estos disturbios obligan a realizar una reparación total ó parcial. El nuevo plan deberá ser generado sin perder de vista que la estabilidad y la eficiencia no deben decrecer.

Los enfoques robustos proactivos son clasificados en tres subcategorías: 1) basados en redundancia; 2) probabilísticos, y 3) con técnicas basadas en contingencias/políticas (Davenport & Beck, 2000). Los objetivos de cada uno son: 1) reducir el impacto de las incertezas y localizar tiempo y recursos extras; 2) obtener las funciones de densidad respecto a las incertezas; y 3) establecer políticas de planificación para atender cualquier secuencia de eventos particular .

Además de las medidas de estabilidad, también es importante conocer cómo los disturbios y las políticas de replanificación afectan la eficiencia del sistema. Los planes robustos permiten

correlacionar el número de disturbios con respecto a la eficiencia del sistema. Se han desarrollado diversos métodos de medición para determinar cómo los eventos estocásticos afectan la eficiencia del sistema. Para referencias véase por ejemplo (Geng et al., 2009) y (Kuster et al., 2010).

Bonfill et al. (2008) propusieron un enfoque estocástico para el modelado y optimización de un problema de replanificación relativa a un proceso de producción por lotes. Inicialmente, se genera un plan proactivo, en el que se incluyen medidas de incerteza acerca de la carga, calentamiento (*heating*) y descarga. Las incertezas fueron caracterizadas por una distribución uniforme. El objetivo de optimización es una combinación del *makespan* y tiempos de espera. El plan es obtenido usando un algoritmo genético optimizado. Siempre que se rompe una máquina ó que los tiempos de procesamiento son más grandes que lo esperado, se replanifica aplicando la regla de desplazamiento a la derecha. Finalmente, los autores desarrollaron un experimento para comparar la eficiencia de los algoritmos bajo enfoques determinísticos y estocásticos. Ellos encontraron que aún y cuando el *makespan* y los tiempos de espera fueron optimizados para un ambiente determinístico, bajo el escenario estocástico, el *makespan* se incrementa por cerca de un 4%. A pesar de la simplicidad del método, los resultados experimentales arrojaron información significativa que apoya el argumento de que es mejor incluir las incertezas desde el momento que inicia el modelado del problema.

Merdan et al. (2011), estudiaron el caso de la aplicación de una política de replanificación manejada por evento a un sistema pequeño de manufactura con falla de máquina como evento disparador de la replanificación. La eficiencia del sistema fue evaluada empíricamente para diferentes tipos de falla y tiempos de terminación de las tareas con diferente duración. La influencia del número de lotes (*pallets*) sobre la eficiencia del sistema también fue evaluada, y se probaron algunas metodologías de replanificación, tales como planificación con desplazamiento a la derecha (*Right-Shift Scheduling*, RS), re-ruteo de agenda (*Agenda Rerouting*, AR), re-ruteo de nuevos trabajos (*New Jobs Rerouting*, NR), re-ruteo completo (*Complete Rerouting*, CR). La evaluación fue implementada en el ambiente de simulación basado en agente, MAST. Los resultados mostraron que CR fue la mejor metodología de replanificación para este caso en particular.

En la investigación reportada en Zakaria and Petrovic (2012), se propuso una replanificación predictiva-reactiva con estrategias de reorganización y no-reorganización para un sistema de manufactura flexible (*Flexible Manufacturing System*, FMS). Los autores consideraron como la única fuente de disturbio la llegada de nuevas órdenes cuando los trabajos programados aún no han

terminado. El reto fue integrar inmediatamente las nuevas órdenes de trabajo al plan de producción existente, mientras que se preserva la estabilidad y la eficiencia de la fábrica. En la estrategia de no-reorganización, las nuevas órdenes son asignadas a las máquinas justo en los tiempos muertos disponibles, mientras que en la estrategia de re-organización, las operaciones son re-secuenciadas para generar una solución parcial dentro del horizonte de la replanificación. Las medidas de eficiencia son un compromiso entre la suma de la tardanza ponderada al cuadrado, el *makespan* y la estabilidad. La estabilidad fue calculada en base a tres aspectos: migración de máquina, tiempo de inicio del trabajo, y desviaciones de la secuencia. Estas estrategias fueron implementadas con GAs. Sus resultados experimentales mostraron que la estrategia de no-reorganización es una mejor opción que la estrategia de reorganización porque la última mejoró la suma de la tardanza ponderada al cuadrado, y al mismo tiempo, la estabilidad fue grandemente incrementada sin incrementar el costo del *makespan*.

Otros investigadores como Novas & Henning (2010), en busca de un balance entre los pros y contras entre los diferentes métodos y políticas, han propuesto un marco que incluye lo mejor de los mundos. El marco de Novas es orientado hacia plantas multi-producto y plantas multi-etapas. Las políticas operacionales son de tres tipos: a) almacenamiento intermedio ilimitado (*Unlimited Intermediate Storage,* UIS); b) no-almacenamiento intermedio, espera ilimitada (*Non-Intermediate Storage, Unlimited-Wait,* NIS-UW); y c) no-almacenamiento intermedio, cero espera (*Non-Intermediate Storage, Zero-Wait,* NIS-ZW). El conocimiento acerca del ambiente de manufactura y el plan de producción es explícitamente representado y modelado con las técnicas orientadas a objeto. Cierta información acerca de los recursos (por ejemplo, acerca de las propiedades y métodos de las entidades más relevantes), se incluye en el conocimiento del dominio. Se consideran también los atributos temporales de los recursos. La política de planificación es manejada por eventos, junto con un método de replanificación parcial. La meta de esta propuesta es dar una solución inmediata a los eventos, sin introducir cambios excesivos en el plan, y al mismo tiempo, mantener la estabilidad del sistema. Cuando ocurre un evento no anticipado, es posible conocer el estado actual de la planificación, gracias a la representación del dominio. Así, en cualquier momento que ocurre un evento, el contexto se obtiene con precisión y es usado para especificar el problema de replanificación. Además, la incorporación de información contextual permite una evaluación precisa del impacto de un evento. Sin embargo, los autores propusieron esta mejora para el futuro. En este caso, la información contextual se usa solamente para evaluar si el plan se ha tornado inútil por efecto de la ocurrencia de algún evento, y en su caso, decidir si se procede la replanificación. Por último, una vez que la replanificación está

completamente especificada y las medidas de eficiencia han sido seleccionadas, se genera el modelo utilizando el método de programación con restricciones (*Constraint Programming*, CP).

El método de Novas (2010) tiene la ventaja de reducir el nerviosismo en la línea de producción, y al mismo tiempo mantener una optimización aceptable. El uso de información contextual demuestra que, cuando se evalúa el impacto del evento exógeno, es posible distinguir el número de los casos donde el tamaño del disturbio hace mandatorio una replanificación, y se evitan así reparaciones innecesarias. Estos resultados experimentales demostraron también, que se obtienen mejores soluciones cuando el dominio del conocimiento incluye secciones más grandes del proceso de manufactura.

Existen también paradigmas emergentes de manufactura que han sido inspirados en biología, por ejemplo, sistemas biónicos de manufactura (*Bionic Manufacturing Systems*, BMS) (Barbosa, 2015), sistemas holónicos de manufactura (*Holonic Manufacturing Systems*, HMS) (Zhao et al.,, 2010) y sistemas reconfigurables de manufactura (Azab & Naderi, 2014). De acuerdo a los paradigmas bio-inspirados, la clave para conseguir la adaptabilidad y la robustez en ambientes cambiantes es la auto-organización. Aun cuando las bases teóricas de los paradigmas bio-inspirados datan de mediados de los 60s, sólo pocas aplicaciones industriales han adaptado este paradigma (Valckenaers et el., 1994), Leitão & Vrba, 2011) y (Mönch et al., 2003).

### 4.3 Conclusiones del Capítulo

Para conducir los sistemas de manufactura hacia una nueva senda en la cual la información fluya suavemente tanto a nivel interno como en su interacción con los demás niveles de producción, es indispensable tomar en cuenta muchos otros parámetros, que en aras de simplificación han sido eliminados de los modelos de planificación. Resulta especialmente importante, tener en consideración los aspectos de incerteza, incompletez de conocimiento y dinamismo que permea el ambiente de producción. La integración de la planificación con los demás niveles de producción, también es uno de los aspectos esenciales que permite la optimización de recursos. Para lograr estas metas, los sistemas de manufactura deberán ser capaces de re-configuración, flexibilidad y robustez. Se anticipa que será necesario utilizar un enfoque híbrido tanto en el etapa de modelado como en la implementación de los futuros sistemas de planificación, en los cuales la representación del conocimiento y los sistemas de razonamiento jugarán un rol destacado.

# Capítulo 5

## PLANIFICACIÓN DE TAREAS DE UN AWS-SMS MEDIANTE ASP

### 5.1 Consideraciones generales

En una actividad orientada a metas en dominios complejos, como lo es el proceso de producción de un sistema de manufactura, típicamente se requiere una combinación de planificación general y planificación de tareas (*scheduling*). El planificador general decide *qué* tareas hay que ejecutar, mientras que la función del planificador de tareas consiste en especificar *cómo* ejecutar el conjunto de acciones, usando un número limitado de recursos y una cantidad limitada de tiempo. Habitualmente, ambas entidades de software son construidas de manera jerárquica, operando estrechamente entrelazados entre sí. En un sistema automatizado, se debe considerar además la estrecha interacción

del planificador de tareas con el controlador del sistema de producción.

Para facilitar el diseño de un planificador, éste puede ser abordado desde diferentes niveles de abstracción. Adicionalmente, el diseño de un planificador habitualmente es estructurado de manera jerárquica siguiendo la estructura lógica en la operación de la empresa que va desde la toma de decisiones a nivel gerencial, ventas y finanzas, hasta el control y operación del sistema de producción. Es importante precisar que el término "los niveles de abstracción" está relacionado con la granularidad, a la cual el mundo es representado, en tanto que la "planificación jerárquica" se asocia con la forma de estructuración del plan, el cual puede o no correlacionarse con "los niveles de abstracción" (Wilkins, 1986).

En un planificador de este tipo, es necesario que la información fluya entre las distintas jerarquías del planificador para que se tomen las mejores decisiones y se optimice el uso de los recursos. La fig. 5.1 muestra un ejemplo de un sistema automatizado vigente en el mercado (VCIM-SCH/Hitachi High-Technologies), donde se aprecia cómo el planificador de tareas utiliza la entrada de datos de diversas fuentes y cómo éste ayuda a controlar el proceso de producción.

Ahora bien, en una actividad orientada a metas en dominios complejos, como lo es un sistema de manufactura, típicamente se requiere una combinación de la planificación general y la planificación de tareas. Lo tremendamente complejo del problema de producción en un SME hace que el desarrollo de un sistema jerárquico de este tipo se encuentre fuera del alcance de esta investigación y es por ello que nos concentraremos exclusivamente en el planificador de tareas en una de las áreas de producción de semiconductores, la más importante, el grabado del circuito en la oblea.

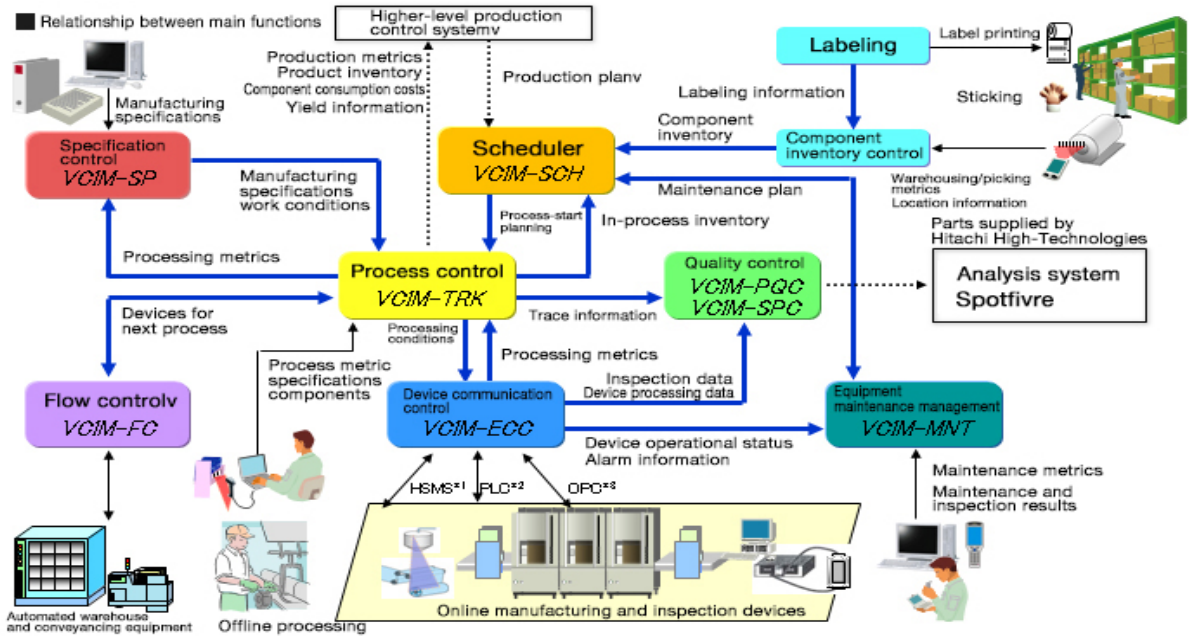



Fig. 5.1 Interacción y flujo de información entre el Planificador de Tareas con otras entidades de producción en el Sistema VCIM-SCH de Hitachi.

Dada la relevancia del problema de planificación de tareas, desde hace varias décadas que investigadores de diversas áreas han enfocado su atención a la resolución de este problema, empleando diversas metodologías y técnicas tanto para el modelado como para la solución, vea por ej. Bhushan & Karimi (2003), Song et al.(2007), Huang et al.(2010), Chen (2011), Katragjini & Ruiz (2013), Lee et al. (2009), Liu et al. (2009), Lombardi & Milano (2012). Sin embargo, una solución general y única persiste como un problema abierto dada su alta complejidad, por lo que resulta válido probar nuevas metodologías para su resolución.

En este capítulo nos enfocaremos precisamente en la descripción, modelado y solución de un problema de planificación de tareas para una sección de un SMS mediante ASP, el cual es un lenguaje para KRR, completamente declarativo, basado en Lógica Autoepistémica, Lógica por Defecto y Lógica de Circumpscripción.

Desde la etapa de diseño hasta la entrega del producto al cliente, existen una serie de procesos que pueden ser mejorados, pero la etapa más importante en la producción de los circuitos integrados (*integrated circuit, IC, CHIP o microchip*) es la de grabado del circuito en la oblea, y es también el área

que representa más retos para reducir los costos y mejorar la satisfacción de cliente, puesto que el costo de la manufactura y la calidad del servicio son significativamente influenciados por el proceso de producción de circuitos (*chips*) debido a su tiempo de producción largo (al menos 30 días y hasta 90 días para algunos tipos de productos) y muy compleja naturaleza operacional (algunos cientos de operaciones ejecutadas por miles de equipos de tecnología-intensiva) (Optesa-white paper).

Básicamente, el proceso de producción de un circuito consiste de las cuatro etapas secuenciales mostradas en la fig. 5.2: 1) la fabricación de obleas, 2) la clasificación por prueba eléctrica, 3) el ensamble y 4) la prueba final. Las primeras dos etapas generalmente son referidas como las etapas frontales (*front-end*) y las dos últimas como terminales (*back-end*).

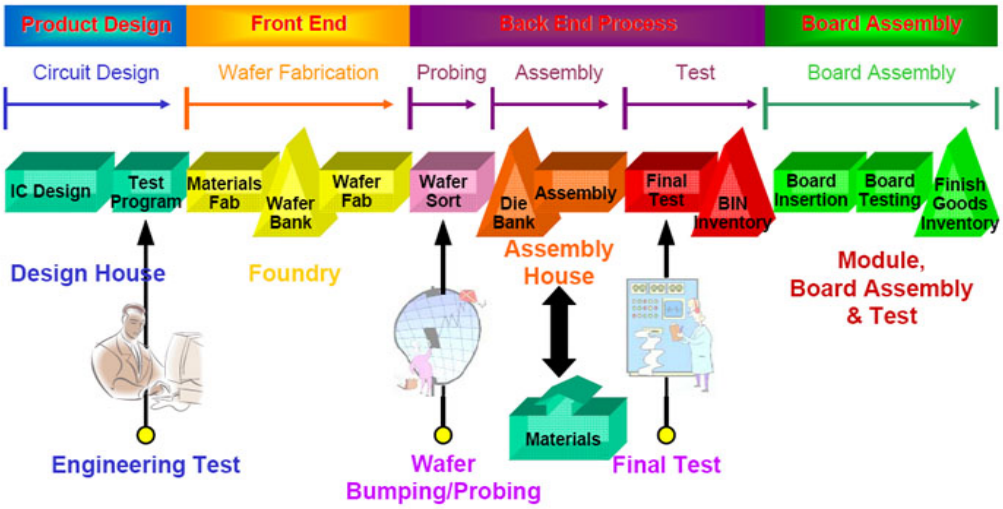



Fig 5.2 Etapas de producción de una fábrica de semiconductores

El desarrollo de un plan óptimo de producción para un SMS, en un horizonte de tiempo con duración desde un día hasta una semana, es un problema extremadamente difícil, puesto que la fabricación de cada oblea consiste de 300 a 500 etapas de procesamiento, con características y/o requerimientos muy exigentes y complejos. Algunas de las características más relevantes de este proceso se describen a continuación y han sido referidas en varias investigaciones, por ejemplo Mönch et al.(2011), Gupta et al. (2006 ):

1. Ambiente no determinístico;
2. Largas líneas de producción

   → Cuello de botella en algunas máquinas.

* Estrategias de control: Evitar *deadlock*, Control de carga;

→ Docenas de flujos de procesos

* Cada flujo de proceso tiene entre 300-900 etapas;

* 100+ máquinas operando en paralelo;

* Procesamiento mezclado de un solo producto ó por lotes;

* Tiempos de ajuste (*setup*) dependientes de la secuencia de trabajos.

3. Concurrencia;

– Flujos reentrantes;

– Múltiples productos en producción al mismo tiempo;

– Duración del proceso en las diferentes etapas desde 15 minutos hasta 12 horas;

– Fallas de herramientas y máquinas;

– Lotes con prioridad;

– Recursos auxiliares compartidos (en situaciones de competencia);

– Procesamiento con ventanas de tiempo;

– Máxima carga de operación de los recursos ;

– Gran número de restricciones en cada proceso de flujo.

Si bien, el problema es complejo, la búsqueda de mejoras en el diseño del planificador no debe ser desdeñado, puesto que un plan más eficiente representa reducción de costos de manufactura y una ganancia más amplia en la venta de los productos. Se ha reportado en plantas pilotos que el uso de nuevas y mejores políticas de planificación redunda en una mejora de hasta un 20% en promedio en tiempos de espera y hasta un 50% en las desviaciones estándar del ciclo de tiempo. También, algunos investigadores han reportado una reducción de costos de hasta $5 millones de dólares en las plantas fabriles, en las cuáles se ha realizado una mejor administración de tiempo y un uso más intensivo de equipo automatizado (Sivakumar, 1999).

## 5.2 Producción de Obleas

La etapa principal en una empresa de semiconductores, es la fabricación de las obleas, y es además un buen punto de partida para mejorar el proceso de producción de un SMS. En esta etapa se construyen los circuitos electrónicos, capa por capa sobre las obleas crudas (delgados discos que se cubren de

óxido de silicio o arseniuro de galio). Este procedimiento involucra cientos de operaciones químicas y físicas que se realizan en diversas etapas.

Es precisamente este proceso de fabricación de las obleas el que resulta de interés para esta investigación, enfocada al modelado y solución del problema de planificación de tareas respecto al proceso de grabado de la oblea, el que generalmente se realiza en un conjunto de estaciones de grabado automatizadas (*Automated Wet-etch Stations, AWS)*, ver fig. 5.3.

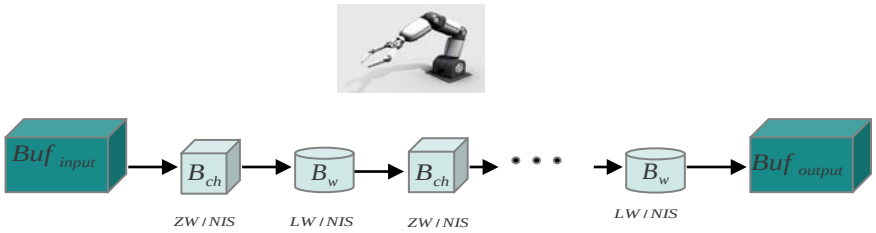



Fig 5.3 Proceso de grabado de circuitos eléctricos en obleas con baños alternos de químicos y agua.

**1.** El proceso de fabricación de una oblea se realiza en lotes y consta de los siguientes procesos y operaciones:

**2.** Se cubre la oblea con una delgada capa uniforme de óxido de silicio ($SiO_2$) ó arseniuro de galio (AsGa).

**3.** Se seleccionan y se marcan porciones de la oblea para formar la configuración del circuito (fotolitografía o fotoenmascaramiento).

**4.** Se aplica el proceso de grabado (*etching*), el cual es la etapa clave en la fabricación de la oblea.

El proceso de grabado se realiza por una o más estaciones altamente automatizadas. En estas estaciones se elimina el film excedente de $SiO_2$ ó AsGa en una serie de baños químicos y deionizantes.

**5.** Los lotes de obleas de la AWS son del mismo tipo y provienen de los procesamientos previos.

**6.** Los lotes son sometidos a procesamiento en los baños químicos y baños de agua, sucesivamente, mediante dispositivos automatizados para el manejo de material, por ejemplo, robots.

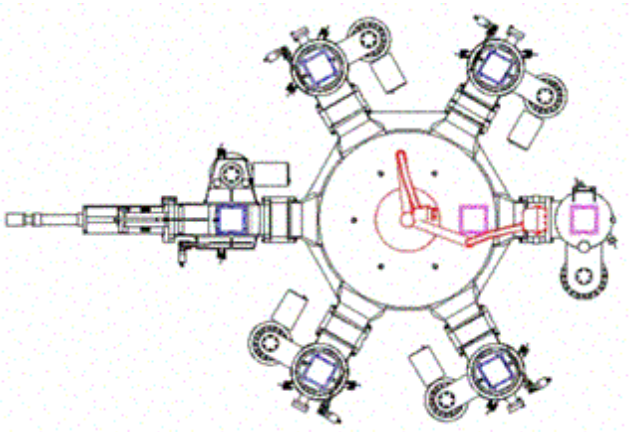
**7.** Los baños químicos son seguidos por una política de almacenamiento de cero espera, los baños de agua son usados como almacenamiento local, lo que en la jerga de planificación de tareas se conoce como *cero espera / no almacenamiento intermedio* (*Zero Wait/No Intermediate Storage, ZW/NIS*) y *almacenamiento local / no almacenamiento intermedio* (*Local Storage/No Intermediate Storage, LS/NIS*).

**8.** El tiempo de procesamiento para un lote dado depende del lote y del baño.

**9.** Un baño solo procesa un lote a la vez.

### 5.3 Descripción Formal

Antes de proceder a realizar la descripción del problema, es importante hacer las siguientes precisiones: el proceso de grabado se realiza en estaciones de trabajo automatizadas y tienen al menos un robot para la transferencia de lotes entre baños. Anteriormente, la disposición de los baños era de forma lineal, y un robot se desplazaba de extremo a extremo para hacer las transferencias de lotes a los baños. Más recientemente, la mayoría de las etapas del proceso de fabricación de semiconductores son realizadas en equipos de una sola pieza, en los cuáles los módulos de procesamiento están dispuestos en forma de racimo e integran uno o más robotos colocados en el centro del equipo. En inglés estos equipos son conocidos como *cluster tools* (ver fig. 5.4). La integración de las etapas en un solo equipo permite mejorar la eficiencia, bajando los costos de producción (Singer 1993). También es cierto que



los *cluster tools* convierten al problema de planificación de tareas en uno con restricciones específicas.

Fig. 5.4 Cluster tools (X-Tronics)

Adicionalmente, es conocido que el problema de Taller de Flujo Robótico es equivalente al

problema de *cluster tool* (Dümmler, 2004). Crama y Van de Klundert (1997) demostraron que la planificación de tareas para un Taller de Flujo Robótico es un problema *NP-complete*.

Aunque el robot tiene un rol importante en el proceso de producción y por ende, es también importante para el diseño del planificador, es importante destacar que en un ambiente real de producción, es difícil modificar la lógica que controla el movimiento del robot que mueve los lotes entre los distintos baños, ya que ésta es fija y no es factible que puede ser modificada porque es propietaria del fabricante del AWS.

Consiguientemente, resulta más conveniente desarrollar una solución independiente de la lógica que controla el robot. Precisamente este es el enfoque que se sigue en esta investigación, donde también se asume que la velocidad del robot es constante y que el robot y todos los componentes de la estación de trabajo están libres de fallas. Considerando estas condiciones, es factible modelar el problema como un problema de Taller de Flujo con *m*-máquinas, con bloqueo en todas las máquinas y condiciones de no-espera en las máquinas que corresponden al baño químico, es decir {1, 3, 5, ..}. Röck (1984) demostró que el problema de taller de flujo con no-espera y *m=3* es *NP-hard* en el sentido estricto. Y como el problema en estudio de esta investigación es más general que el problema de *m=3*, por lo tanto, también es *NP-hard*. De acuerdo a la notación de Graham et al. (1979) y Brucker (1999), el problema es denotado por

*Fm | no-pmtm, permutation, no-wait | Cmin.*

### 5.3.1 Asunciones del Modelo

Dado un conjunto de máquinas de un Sistema Automatizado de Grabado (AWS) de una empresa de semiconductores y un conjunto de trabajos con su duración especificada, el problema consiste en realizar la planificación de tareas y la optimización del plan. La función objetivo es la minimización del *makespan*. Todos los trabajos deben ser procesados secuencialmente en todas las máquinas, iniciando siempre en la primer máquina y terminando en la última. Los trabajos no se interrumpen. Las máquinas representan un conjunto de baños químicos y baños de agua dispuestos alternadamente. El plan resultante tiene que cumplir con restricciones estrictas en el sentido de que una vez que un trabajo termine su procesamiento en un baño químico, inmediatamente después debe ser asignado al siguiente baño de agua. Los baños de agua fungen también como almacenamiento temporal hasta que se libere el siguiente baño químico. Se asume que los baños no requieren de ningún tipo de reinicio (*setup*) y que no hay rupturas asociadas con éstos.

Aunque en sistemas reales de manufactura, los lotes son movidos entre los baños por uno o más robots, en esta investigación se simplifica el problema, considerando que la velocidad del robot es constante y los tiempos de traslado son tan pequeños en relación con la duración de los procesos, que es despreciable y no tomada en cuenta. Además, dado que los tiempos de operación de estos robots son del orden de milisegundos, y el lenguaje ASP, con el cual se resolverá el problema, solo opera con números enteros, por lo tanto no es posible diseñar programas ASP que incorporen información que requiera el manejo de números reales.

Dadas las características y restricciones, el problema de planificación consiste en determinar la secuencia de lote de obleas en cada baño. La meta es la minimización del *makespan*. Además, el problema puede ser descrito como un Problema de Planificación de Estaciones de Taller de Flujo Seriales Multiproducto con Políticas ZW/NIS y LS/NIS. A continuación se hace una descripción formal del problema.

### 5.3.2 Modelado del Problema

**Nomenclatura**

*Índices*

*j* - lote (trabajo);

*k* – baño.

*Conjuntos*

*J* - conjunto de lotes de obleas, con *J=1, . . , n*

*R* - conjunto de *k* recursos renovables (*k* es el número preestablecido de unidades de un recurso que está disponible para cada período del horizonte de planificación T). En este caso R es el conjunto de baños. La disposición de los baños es alternada, iniciando por un baño químico, seguida de un baño de agua, donde R = $\{B_{ch}, B_w\}$. Además, $B_{ch}$ es a su vez el conjunto de baños químicos descritos por $B_{ch} = \{1,3,5,\ldots,k\text{-}1\}$. El conjunto de los baños de agua está dado por $B_w = \{2,4,6,\ldots, k\}$.

*D* - conjunto de las especificaciones de procesamiento de los trabajos. Cada $d \in D$ es una tupla

formada por (A, M, P ), donde

1. A es un número identificando al trabajo *j*;
2. M es un número identificando al baño *k* asignado para procesar el trabajo *j*;
3. P es un número identificando el tiempo $p_{jk}$ de residencia del trabajo *j* en el baño *k*.

*T* - conjunto de rangos de valores de tiempo, para los cuales debe ajustarse el plan, dado por

T = {1,…, *deadline*}.

*U* - conjunto de los tiempos de inicio de procesamiento de los trabajos y cada $u \in$ U es una tupla formada por (A, M, T, P ), donde

1. A es un número identificando al trabajo *j*;
2. M es un número identificando al baño *k*, asignado para procesar el trabajo *j*;
3. T es el tiempo $t_{jk}$ de inicio del procesamiento del trabajo *j* en el baño *k*;
4. P es un número identificando al tiempo $p_{jk}$ de residencia del trabajo *j* en el baño *k*.

*S* - conjunto $S = \{S_1,\ldots, S_n\}$, que representa un plan.

*A* - conjunto que relaciona los trabajos asignados a los baños, incluyendo los tiempos de inicio y la duración de los trabajos, cada $a \in$ A es una tupla formada por (N, M, T, P), donde

- N es un número identificando al índice del trabajo *j*;
- M es un número que identifica al índice del baño *k* asignado para procesar el trabajo *j*.
- T es el tiempo $t_{jk}$ de inicio del procesamiento del trabajo *j* en el baño *k*;
- P es un número que identifica el tiempo $p_{jk}$ de residencia del trabajo *j* en el baño *k*.

*Parametros y constantes*

*deadline* - tiempo máximo de terminación de todos los trabajos, una constante entera *deadline* > 0.

$n_b$ - número de baños de cierta instancia, una constante entera $n_b > 0$.

$n_j$ - el número de trabajos de una instancia dada, una constante entera $n_j > 0$.

### 5.3.3 Funciones y Restricciones

Para hacer una descripción formal de este problema, se introducen las siguientes funciones para definir las restricciones matemáticas que operan sobre el dominio del problema. Estas descripciones nos

permiten especificar el problema de una forma precisa, matemáticamente hablando, y con ello restringir el dominio de entrada descrita en el la sección 5.2. Adicionalmente, estas funciones establecen una definición formal de conceptos informales descritos en la sección 5.1 de este mismo capítulo.

**Definición 5.1** Definimos `badSeq(α,β)` para alguna $\alpha_{AMTP} \in U$ distinta de $\beta_{AMT} \in U$ como una función que cumple con las siguientes restricciones:

$\alpha_A = (j_\alpha), \beta_{\mathcal{A}} = (j_\beta) \rightarrow j_\beta = j_\alpha,$

$\alpha_{\mathcal{M}} = (k_\alpha), \beta_{\mathcal{M}} = (k_\beta) \rightarrow k_\beta = k_\alpha + 1,$

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}), \beta_T = (t_{\beta 0}) \rightarrow t_{\beta 0} < t_{\alpha 1}.$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$

**Definición 5.2** Definimos `anyJobStartAtOne(α,β)` para alguna $\alpha \in U$ distinta de $\beta \in U$ como una función que regresa valor verdadero si y solo si

$min(\alpha_{AMTP}, \beta_{AMTP}) > 1,$

ó bien

$\alpha_A = (j_\alpha), \beta_A = (j_\beta) \rightarrow j_\beta \neq j_\alpha,$

$\alpha_M = (k_\alpha), \beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha = 1,$

$\alpha_T = (t_{\alpha 0}), \beta_T = (t_{\beta 0}) \rightarrow \min(t_{\alpha 0}, t_{\beta 0}) > 1, \forall j_\alpha, j_\beta$ □□□

**Definición 5.3** Definimos `badSincronyBch(α,β)` para alguna $\alpha \in U$ distinta de $\beta \in U$ como una función que regresa valor verdadero si y solo si $(\alpha_{AMTP}, \beta_{AMTP})$, cumple con las condiciones

$\alpha_A = (j_\alpha), \beta_A = (j_\beta) \rightarrow j_\beta = j_\alpha$

$\alpha_M = (k_\alpha), \beta_M = (k_\beta) \rightarrow (k_\alpha \ mod\ 2 > 0) \wedge (k_\beta = k_\alpha + 1)$

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}), \beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow t_{\beta 0} > t_{\alpha 1}.$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y $t_{\beta 1} = t_{\beta 0} + p_{\beta 0}$

**Definición 5.4.** Definimos `overlapBathJobs(α,β)` para alguna $\alpha \in U$ distinta de $\beta \in U$,

como una función que regresa valor verdadero si y solo si $\alpha_{AMTP} \cap \beta_{AMTP} = 1$, ó bien

$\alpha_A = (j_\alpha)$, $\beta_A = (j_\beta)$, $\rightarrow j_\beta \neq j_\alpha$,

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha$,

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1})$, $(t_{\alpha 0} \leq t_{\beta 0}) \wedge (t_{\alpha 1} > t_{\beta 0})$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y $t_{\beta 1} = t_{\beta 0} + p_{\beta 0}$

**Definición 5.5.** Definimos `overlapJobBaths(α,β)` como una función para alguna $\alpha \in U$ distinta de $\beta \in U$, de la siguiente forma $\alpha_{AMTP} \cap \beta_{AMTP} = 1$, ó bien

$\alpha_A = (j_\alpha)$, $\beta_A = (j_\beta) \rightarrow j_\beta = j_\alpha$,

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha + 1$,

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow (t_{\alpha 0} \leq t_{\beta 0}) \wedge (t_{\alpha 1} > t_{\beta 0})$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y $t_{\beta 1} = t_{\beta 0} + p_{\beta 0}$

Antes de definir las siguientes restricciones, es pertinente aclarar que fue necesario considerar que no debe haber demasiado retraso en el inicio de procesamiento de los diferentes trabajos. Aunque inicialmente se intentó usar una restricción estricta en el sentido de que inmediatamente que un trabajo termine en el primer baño, debe asignarse a otro de los trabajos pendientes. Esta restricción resulta contraproducente, porque para la instancia del problema que se usó en esta investigación, Zeballos (2011) encontró que la solución óptima es áquella en la cual existe retraso en la asignación del primer baño. Estos resultados también fueron confirmados con los resultados experimentales realizados en esta investigación, ya que si se aplica esta restricción no existe solución para trabajos más grandes de 4 trabajos y 4 baños.

Entonces, se decidió flexibilizar la restricción, permitiendo cierto retraso en el uso del primer baño entre distintos trabajos. Para calcular el valor permitido, se aplicó la heurística de que el máximo retraso entre el inicio de cualquiera dos trabajos, se calcula en base un parámetro introducido a tiempo de ejecución. Una opción adecuada para este parámetro es la diferencia entre el trabajo más largo y la fecha límite de ejecución (*deadline*). Esta restricción se define como sigue:

**Definición 5.6.** Definimos `badSincInB1s(α,β)` para alguna $\alpha \in U$ distinta de $\beta \in U$ como

una función que regresa valor verdadero si y solo si ($\alpha_{AMTP}$ , $\beta_{AMTP}$ ) cumple con las condiciones:

$\alpha_A = (j_\alpha)$, $\beta_A = (j_\beta) \rightarrow j_\beta \neq j_\alpha$,

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha$,

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow ( t_{\beta 0} = \lceil t_{\alpha 0} \rceil ) \wedge (t_{\alpha 1} > t_{\beta 0}) \wedge (t_{\beta 0} - t_{\alpha 1}) >$ `maxDelaysB1`

donde

$t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y `maxDelaysB1`

en esta primera aproximación es un valor que especifica el usuario al momento de ejecución.

Las funciones que se describen a continuación forman parte de las funciones dedicadas a establecer cuándo se debe hacer la asignación de un trabajo a un baño y detectar condiciones de conflicto entre trabajos, ya que las funciones definidas de 5.1 a 5.5 solo verifican condiciones de asignación de baños entre un solo trabajo.

Las definiciones 5.7 a la 5.10 se enfocan en definir las funciones para hacer la asignación de trabajos a los recursos. En cada una de estas funciones se consideran diferentes situaciones que se deben tener en cuenta para determinar si procede la asignación.

La primera función lidia con el caso en que no ha terminado ningún trabajo:

**Definición 5.7.** Definimos `assigned(`$\alpha$`)` para alguna $\alpha \in U$ distinta de $\beta \in U$ como una función que regresa valor verdadero si y solo si ($\alpha_{AMTP}$ , $\beta_{AMTP}$ ) cumple con las condiciones:

$\alpha_A = (j_\alpha)$, $\beta_A = (j_\beta) \rightarrow j_\beta = j_\alpha$,

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha + 1$,

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow (t_{\alpha 0} < t_{\beta 0}) \wedge (t_{\beta 0} > t_{\alpha 1})$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ .

La segunda función que tiene que ver con la asignación de recursos, considera el caso en que un trabajo dado solo puede ser asignado al siguiente baño si ya terminó en el anterior

**Definición 5.8.** Definimos `assigned(`$\alpha$`)` para alguna $\alpha \in U$ como una función que devuelve valor verdadero si y solo si ($\alpha_{AMTP}$) y ($\beta_{NM(T+P)}$ ) cumplen con las condiciones que se detallan a

continuación: .

NOTA: Esta función requiere de los valores devueltos por una función que se define más adelante nombrada `finished(β)`, donde $\beta \in A$

$\alpha_A = (j_\alpha)$, $\beta_N = (j_\beta) \rightarrow j_\beta = j_\alpha$,

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow k_\beta = k_\alpha + 1$

$\alpha_T = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow t_{\beta 0} \geq t_{\alpha 1}$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ .

La tercera y última función que tiene que ver con la asignación de recursos, considera el caso en que un trabajo dado solo puede ser asignado al siguiente baño si ya terminó de procesarse en el anterior, y no hay razón para creer que ya se haya procesado antes en dicho baño.

**Definición 5.9.** Definimos `assigned(α)` para alguna $\alpha \in U$ como una función en los términos que se anotan en el siguiente párrafo.

NOTA: Esta función requiere de los valores devueltos por una función que se definen más adelante nombradas `finished(β)`, y `finished(γ)`, donde $\beta \in A$ y $\gamma \in A$ y $\beta \neq \gamma$. La función `assigned(α)` devuelve valor verdadero si y solo si ($\alpha_{AMTP}$ ), $\neg(\beta_{NMTP}$ *) y* ($\gamma_{NMTP}$ ) cumplen con las condiciones que se detallan a continuación:

$\alpha_A = (j_\alpha)$, $\beta_N = (j_\beta)$, $\gamma_N = (j_\gamma) \rightarrow (j_\beta = j_\alpha) \wedge (j_\gamma \neq j_\alpha)$

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta)$, $\gamma_M = (k_\gamma) \rightarrow k_\beta = k_\alpha = k_\gamma$

$\alpha_T = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1})$, $\gamma_{TP} = (t_{\gamma 0}, t_{\gamma 1}) \rightarrow (t_{\alpha 0} \geq t_{\gamma 1}) \wedge (t_{\beta 1} \leq t_{\alpha 0})$

donde $t_{\beta 1} = t_{\beta 0} + p_{\beta 0}$ y $t_{\gamma 1} = t_{\gamma 0} + p_{\gamma 0}$ .

**Definición 5.10.** Dado una cierta $\delta \in A$ y otra diferente $\alpha_{NMTP} \in A$, definimos la función `finished(δNMTP )` como sigue:

$\alpha_N = (j_\alpha)$, $\delta_N = (j_\delta) \rightarrow j_\delta = j_\alpha$

$\alpha_M = (k_\alpha)$, $\delta_M = (k_\delta) \rightarrow k_\delta = k_\alpha$

$\alpha_T = (t_{\alpha 0}, t_{\alpha 1})$, $\delta_T = (t_{\delta 0}, t_{\delta 1}) \rightarrow (t_{\alpha 0} < t_{\delta 1})$

y además $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y $t_{\delta 1} = t_{\delta 0} + p_{\delta 0}$.

**Definición 5.11**. Cuando se trata de asignar un trabajo a un baño, es necesario determinar si el baño no está ocupado por otro trabajo, para una cierta $\alpha \in A$ y otras diferentes $\beta \in A$ y $\delta \in A$, se define la función `busyBath(β)` mediante una función booleana que devuelve verdadero si y solo si $(\beta_{NMTP} \cap \delta_{NMTP}) = 1$, ó

$\alpha_N = (j_\alpha)$, $\beta_N = (j_\beta)$, $\delta_N = (j_\delta) \rightarrow (j_\alpha = j_\beta) \wedge (j_\beta \neq j_\delta)$

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow (k_\alpha \bmod 2 = 0) \wedge (k_\beta = k_\alpha + 1)$

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1})$, $\delta_{TP} = (t_{\delta 0}, t_{\delta 1}) \rightarrow (t_{\beta 0} > t_{\alpha 1}) \wedge (t_{\delta 1} \leq t_{\alpha 1}) \wedge (_{max}(t_{\delta 0}) = t_{\beta 0}, \forall_{\delta \in A})$

$\wedge (_{max}(t_{\delta 0}) > t_{\alpha 1}, \forall_{\delta \in A})$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ y $t_{\delta 1} = t_{\delta 0} + p_{\delta 0}$.

**Definición 5.12** Cuando un trabajo se mueve de un baño químico a un baño de agua, la transferencia debe ser inmediata; esta función define cómo verificar aquéllos trabajos que hayan sido asignados de forma no inmediata o retrasados para una cierta $\alpha_{NMTP} \in A$ u otra diferente $\beta_{NMTP} \in A$, se define la función `delayedBw(α)` mediante una función booleana que devuelve valor verdadero si y solo si $\alpha_{AMTP} \cap \beta_{MTP} = 1$, ó

$\alpha_N = (j_\alpha)$, $\beta_N = (j_\beta) \rightarrow j_\alpha = j_\beta$

$\alpha_M = (k_\alpha)$, $\beta_M = (k_\beta) \rightarrow (k_\alpha \bmod 2 = 0) \wedge (k_\beta = k_\alpha + 1)$

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$, $\beta_{TP} = (t_{\beta 0}, t_{\beta 1}) \rightarrow (t_{\beta 0} > t_{\alpha 1}) \wedge$ `not ⌐delayedBw(β),`

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ .

**Definición 5.13.** Una de las etapas para obtener el mínimo *makespan*, es determinar cual es el tiempo máximo que tarda en terminar cada trabajo en toda $\alpha_{NMTP} \in A$; para ello definimos el conjunto $\delta$ que contiene los tiempos máximos de terminación de todos los trabajos. Estos tiempos máximos se obtienen mediante la función `lastJob(`$\alpha_{TP}$`)`

$\alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1})$

$\delta = \max(t_{\alpha 1}), \forall_{\alpha \in A}$

donde $t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$ .

### 5.3.4 Funciones para Cálculo de Rangos

Las funciones que se definen a continuación se refieren a una mejora que se hizo al programa para que se reduzca el número de combinaciones que se generan durante la etapa de instanciación (*grounding*) en ASP. Básicamente el enfoque consiste en obligar a que el número de combinaciones por trabajo y baño se reduzca a un cierto rango de tiempo, en lugar de que se genere para todo el período que comprende, desde el tiempo inicial hasta la fecha límite. Este método basado en rangos tiene sentido por el tipo de problema de planificación de tareas en estudio, o sea, el modelo de Taller de Flujo. Es decir, todos los trabajos necesariamente deben procesarse de manera secuencial empezando siempre en el primer recurso. En consecuencia, es irrelevante la generación de tiempos de inicio que incluyan el rango completo. La ventaja de este método basado en rangos, es que se reduce grandemente el espacio de búsqueda y con ello el problema se convierte en un problema tratable para instancias de mayor tamaño.

**Definición 5.14**. En primer término, necesitamos calcular la sumatoria de los tiempos de procesamiento de cada trabajo en cada baño. El conjunto $\lambda$ contiene estas sumatorias, que es el valor devuelto por la función `dBsumOne(α,λ)` para cada $\alpha_{NMTP} \in A$ :

$$\lambda = \sum_{k_{\alpha=1}}^{nb} t_{\alpha 1}, \quad \forall j_\alpha$$

donde el número de baños está dado por *nb* y

$$\alpha_N = (j_\alpha), \alpha_M = (k_\alpha), \alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}) \rightarrow t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$$

**Definición 5.15** Después, con el valor calculado en la función definida en 5.14, determinamos el valor de máximo de retraso que se debe permitir a cada trabajo, mediante la función `maxDelayAnyJ(α,μ)`, para cada $\alpha_{NMTP} \in A$. La función devuelve los resultados calculados en el conjunto que denominamos $\chi$ ,

$$\chi = \sum_{j_\alpha=1}^{np} deadline - \lambda_{j\alpha}$$

donde el número de trabajos está dado por *nj*; *deadline* es la fecha límite y

$\alpha_N = (j_\alpha), \alpha_M = (k_\alpha), \alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}) \rightarrow t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$

**Definición 5.16**. Para calcular el rango en que se deben generar las combinaciones de cada baño para un cierto trabajo, necesitamos conocer, cual es la sumatoria de los tiempos de procesamiento desde el primer hasta el *x-ésimo baño*, donde el *x-ésimo* baño es aquel previo al baño, para el cual se está calculando el rango. La función que define cómo hacer este cálculo es nombrada `dSumJuntilB(α, ξ, ρ)` para cada $\alpha_{NMTP} \in A$. El valor es devuelto en el conjunto ρ

$$\rho = \sum_{k\alpha=1}^{\zeta\alpha} t_{\alpha 1} \qquad \forall_{j\alpha}$$

$\alpha_N = (j_\alpha), \alpha_M = (k_\alpha), \alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}) \rightarrow t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$

**Definición 5.17**. Por último, definimos la función `rango(α, Ωupper, Ωlower)`, donde se determina cómo obtener el límite superior, `Ωupper`, y el límite inferior, `Ωlower`, del rango de tiempo de inicio. Para calcular estos valores, deben generarse las combinaciones para cada trabajo *j* y cada baño *k* en cada $\alpha_{NMTP} \in A$, de acuerdo a la siguiente fórmula:

$$\Omega_{upper} = \chi_{j\alpha} + \rho_{j\alpha} - t_{\alpha 1_{j\alpha, k\alpha}} \qquad \forall_{j\alpha} \quad \forall_{k\alpha}$$

donde

$\alpha_N = (j_\alpha), \alpha_M = (k_\alpha), \alpha_{TP} = (t_{\alpha 0}, t_{\alpha 1}) \rightarrow t_{\alpha 1} = t_{\alpha 0} + p_{\alpha 0}$

El límite inferior del rango de cada baño *k* y cada trabajo *j* es devuelto en el conjunto `Ωlower` según se define en la siguiente ecuación:

$$\Omega_{lower} = \Omega_{upper_{j\alpha, k\alpha}} - \chi_{j\alpha} + 1 \qquad \forall j_\alpha \quad \forall k_\alpha$$

donde

$\alpha_N = (j_\alpha), \alpha_M = (k_\alpha)$

## 5.4 Representación del Modelo en ASP

El objetivo principal de esta investigación es encontrar un plan optimizado para el problema AWS de

una fábrica de semiconductores. La optimización consiste en la minimización del *makespan* resultante. Para la obtención del plan optimizado se usa el lenguaje de representación de conocimiento y razonamiento, ASP, que es tecnología de punta, y diseñada especialmente para la solución de problemas combinatorios difíciles (como lo es el problema de planificación de tareas) (Gebser et al., 2015) .

Los problemas de planificación de tareas y optimización cuando son modelados con ASP, requieren el uso de variables que representan los posibles valores de la función objetivo, que a menudo forman dominios numéricos extremadamente grandes. Por ejemplo, tanto el problema de planificación como el de planificación de tareas (*scheduling*), requieren que los programas de *answer set* incluyan variables representando los tiempos, en los cuales ocurren los eventos planificados (por ejemplo, debe haber una combinación formada por al menos tres datos: cada una de las unidades de tiempo durante el horizonte de sucesos, cada trabajo y cada máquina. El número de combinaciones resultante es prohibitivamente grande, especialmente cuando los trabajos requieren muchas unidades de procesamiento, puesto que este es el principal factor que incide en la explosión combinatoria del problema. Cuando esto ocurre, la etapa de instanciación es una tarea computacional muy pesada, en el mejor de los casos. En el caso peor, la tarea es un problema intratable.

Una metodología común para resolver un problema en ASP es diseñar el programa en dos partes: **GENERA** y **PRUEBA** (Lifschitz, 2001). La primera parte define la colección de *answer sets* que representan soluciones potenciales. La segunda consiste de reglas que eliminan los *answer sets* que no son soluciones. A menudo se incluye una tercera parte en el programa, la sección DEFINE. En esta última sección se expresan conceptos auxiliares que son usados para codificar las restricciones. Así, cuando se representa un problema en ASP, existen dos clases de reglas que tienen un rol especial: las que generan tantas soluciones como sea posible, y las que se utilizan para eliminar los *answer sets* que no corresponden a las soluciones.

### 5.4.1 ASP: Metodología Genera y Prueba

En esta primera parte se presenta la codificación del problema en ASP para ser resuelto con la máquina de inferencias Clingo versión 4.5 (Gebser et al., 2014). La codificación del problema se hace de forma completamente declarativa, siguiendo la descripción formal del problema. La sección **GENERA** sigue el formato regular aconsejado para la generación de todas las posibles combinaciones. Después se describe una codificación alternativa, en la cual la sección de **GENERA** ha sido modificada para evitar

la generación de todas las posibles combinaciones, puesto que la función de generación es una función exponencial que depende fuertemente del tamaño de las unidades de tiempo que se necesitan para procesar todos los trabajos, como se detallará más adelante. La explosión combinatoria inherente al problema tiene como consecuencia que aún y cuando las instancias no sean demasiado grandes, el problema se vuelve intratable. Entonces, explotando el conocimiento del problema, se definen los casos que no representan soluciones para este problema en particular, y se incluye este conocimiento en la regla que genera las combinaciones a fin de que se reduzca el espacio de búsqueda, y con el objetivo de resolver instancias más grandes del problema.

### 5.4.2 Codificación: Primer Enfoque

Utilizando el enfoque de ASP completamente declarativo para resolver el problema de planificación de tareas para un AWS, la meta es codificar el problema como un programa lógico de forma que los *answer sets* del programa correspondan a las secuencias del orden en que los lotes deben ser planificados a fin de que el *makespan* sea mínimo.

A continuación se presenta la codificación del problema en ASP. Se asume que *nb* y *nj* son enteros que nos dan el número de baños (recursos) y el número de lotes, respectivamente. Estas constantes se definen en el programa como los siguientes hechos:

```
bath(1..nb).  numBaths(nb).
```
(5.4.1)

```
job(1..nj).  numJobs(nj).
```
(5.4.2)

El tiempo máximo para que cualquier trabajo termine, o sea el *deadline,* se especifica con el hecho:

```
times(1..deadline).
```
(5.4.3)

Los enteros *nb, nj* y *deadline* son parámetros que se introducen al momento de ejecución.

El tiempo de procesamiento requerido para cada trabajo y cada máquina, está especificado por hechos de la forma:

*duration*($j$, $k$, $p_{jk}$),

donde $j$ es un entero que toma valores dentro del rango (1, .., *nj*), y representa un identificador para un trabajo; $k$ también es un entero con rango (1, .., *nb*) que representa un identificador para cada baño; por

último, $p_{jk}$ representa el tiempo de procesamiento.

En la sección **GENERA**, se establece la siguiente regla que nos permite generar todas las posibles combinaciones de trabajos, baños, tiempos de inicio (desde 1 hasta *deadline)* y tiempos de procesamiento de los trabajos. Cada posible solución es un cierto número de secuencias del tipo `do(J,B,T,D)`, donde J es el número de trabajo, B es el número de baño, T es el tiempo de inicio de J en B, y D es el tiempo de procesamiento de J en dicho baño:

```
{do(J,B,T,D) : times(L), T=1..L }=1 ← duration(J,B,D),          (5.4.4)
job(W),J=1..W, bath(M), B=1..M.
```

Por ejemplo, para una instancia del problema que contiene 2 trabajos, 2 baños y una fecha límite de 18, un fragmento de estas secuencias es la siguiente

```
Answer: 1 → do(1,1,1,4) do(1,2,12,6) do(2,1,1,6) do(2,2,11,7)
Answer: 2 → do(1,1,1,4) do(1,2,12,6) do(2,1,2,6) do(2,2,11,7)
Answer: 3 → do(1,1,1,4) do(1,2,12,6) do(2,1,3,6) do(2,2,11,7)
Answer: 4 → do(1,1,2,4) do(1,2,12,6) do(2,1,1,6) do(2,2,11,7)
Answer: 5 → do(1,1,2,4) do(1,2,12,6) do(2,1,2,6) do(2,2,11,7)
```

Adicionalmente, en la parte **DEFINE** se especifican reglas de dos tipos: En el primer grupo se incluyen las reglas que eliminan las secuencias inválidas `do(J,B,T,D)`generadas previamente mediante la regla indicada en 5.4.4. Estas reglas auxiliares nos permiten determinar con precisión, cuáles de las secuencias generadas no reúnen las restricciones acerca del orden en que deben ser secuenciados los trabajos y también la restricción estricta 'cero espera' (ZW) en la asignación entre cada baño químico y su sucesor baño de agua. Después, a partir de las secuencias resultantes de la reducción realizada con las reglas antes descritas, se determina, cuándo un trabajo debe ser asignado a cada uno de los baños. Las reglas 5.4.5 - 5.4.10 pertenecen al primer grupo. El siguiente grupo de reglas está dedicado a definir las condiciones bajo las cuáles es válida la asignación.

En la regla que se describe a continuación se implementa la función 5.1. Primeramente, nos tenemos que asegurar que la secuencia generada `do(J,B',T',D')`siga la restricción de procesamiento del trabajo en baños secuenciales, verificando que para todo trabajo J, el tiempo de inicio T' en B', es posterior al procesamiento en el baño previo, `do(J,B,T,D).` O sea, que el

procesamiento del trabajo J en el baño B, preceda al procesamiento de J en B'. Además B' debe ser el baño que sucede inmediatamente a B. Si no cumple con la restricción de secuenciamiento, la regla se dispara:

```
badSeq(J,B,T,D,B',T',D') ← times(T'), do(J,B',T',D'),          (5.4.5)
    do(J,B,T,D), B'==B+1, T'<T+D.
```

La regla 5.4.6 implementa la función definida en 5.2. El objetivo de esta función es el de asegurar que dentro de la secuencia generada haya al menos un trabajo que inicia en el primer baño B al tiempo T = 1. Esto fue implementado usando el agregado *#min* de Clingo, el cual nos devuelve en la variable *Mn* el tiempo mínimo, *T*, de todas las combinaciones en cada secuencia. Si *Mn > 1*, significa que no hay ningún trabajo iniciando en el primer baño con T = 1, entonces esta solución debe ser eliminada. La regla en cuestión es la siguiente:

```
anyJobStartAtOne(Mn)← Mn=#min{T: times(T), times(T'),          (5.4.6)
    do(J,B,T,D), do(J',B,T',D'), J != J',
    B==1,T' > T}, Mn > 1.
```

Para implementar la restricción relativa a la secuenciación inmediata entre baños químicos y baños de agua, definida en la función 5.3, se hace uso de la secuencia `do(J,B,T,D)`que establece el caso de si un trabajo cualquiera J que se haya procesado en un baño B impar, y no sea asignado inmediatamente después al siguiente baño B' de agua, según `do(J,B',T',D').` Cuando esto ocurre, el *answer set* respectivo debe eliminarse

```
badSincronyBch(T')← do(J,B,T,D), do(J,B',T',D'),               (5.4.7)
    B\2 > 0, B' ==B+1, T' > T+D.
```

El posible traslape es de dos tipos: a) los trabajos J y J', donde J ≠ J' se traslapan en el mismo baño, y b) el mismo trabajo J aparece asignado a dos baños secuenciales, B y B', en tiempos traslapados. El primer caso se refiere a la función definida en 5.4 y el segundo caso a la definida en 5.5:

```
overlapBathJobs(T') ← do(J,B,T,D), do(J',B,T',D'),             (5.4.8)
bath(B), J != J', T <= T', T + D > T'.
overlapJobBaths(T') ← do(J,B,T,D), do(J,B',T',D'),             (5.4.9)
    B'==B+1, T<=T', T+D > T'.
```

La regla 5.4.10 implementa la función definida en 5.6, la cual detecte violaciones a la restricción

de que todos los trabajos deben iniciarse en el primer baño, dentro de un cierto margen máximo después de que termine el último trabajo procesándose en ese baño. En la implementación, se busca cual es el tiempo X de inicio de cada trabajo en el baño inicial B y se determina cual otro trabajo J' o J'' tiene el tiempo sucesor Z de X. La búsqueda del trabajo sucesor de J en el primer baño se implementó con una condición compuesta en el cuerpo de la regla, expresada por *(#false* : `X<Y, do(J',B,Y,D'), Y<Z;`). Esta condición evalúa a falso si hay un elemento Y entre X y Z, obligando a que sean descartadas las instancias, donde X y Z no son sucesores directos, porque el cuerpo de la regla es falso. En cambio, cuando X y Z son sucesores directos uno del otro, la condición es falsa para todos los Y elementos. Después solo hay que probar si el tiempo en que termina el trabajo J, F=X+D, es mayor que el del trabajo sucesor Z, y probar si la diferencia entre ambos es mayor que el parámetro estipulado como máximo retraso permitido, según se nota en la siguiente regla:

```
badSincInB1s(J,X,F,J'',Z) ← B==1, do(J,B,X,D),                    (5.4.10)
    #false :   X < Y, do(J',B,Y,D''), Y < Z; do(J'',B,Z,D''),
    X < Z, F=X+D, Z>F, L=Z-F, L>maxDelaysB1.
```

Las reglas que se describen a continuación forman parte del segundo grupo de reglas antes mencionado. La estrategia que se siguió para determinar la asignación de baños a los trabajos, fue la de verificar el secuenciamiento y la restricción ZW. Si bien estas primeras reglas del 5.4.11 a 5.4.13, que implementan las funciones de 5.7 a 5.9, verifican condiciones de asignación entre un solo trabajo.

Las reglas de 5.4.11 a 5.4.13 especifican cuándo un trabajo debe asignarse a cada uno de los baños. Las reglas establecen que los trabajos deben ser procesados en los baños en orden secuencial estricto, empezando siempre por el primer baño. También se tiene que cumplir con otra restricción estricta: en cuanto un trabajo termine de procesarse en un baño químico, éste debe ser asignado inmediatamente (*zero wait*, ZW) al baño de agua siguiente. Además, para que la asignación sea válida, también el tiempo de procesamiento de la asignación debe ser menor o igual a la fecha límite. Por eso, para fijar estas restricciones, fue necesario definir una regla para cada uno de los tres casos siguientes (todas, además, verifican que la asignación esté en tiempo):

1) La regla *assigned* identificada con el número 5.4.11, se usa para hacer la asignación del primer trabajo. Es decir, en este punto aún no hay ningún trabajo que haya terminado de procesarse. El `assigned(J,B,T,D)`que aparece en la cabeza de la regla 5.4.11 indica que el trabajo J es asignado al

baño B, con tiempo de inicio T y tiempo de residencia D. Esto ocurre si para las secuencias `assigned(J,B,T,D)` y `do(J,B',T',D'),` existe un mismo trabajo J, que en cada secuencia tiene baños diferentes B y B' con tiempos de inicio T y T', y duraciones D y D'; además se cumplen las condiciones: B' es el baño que sucede a B y T' ocurre después de que J termine de procesarse en B porque T' > = T+D.

2) La regla 5.4.12 se activa cuando ya se tiene terminado algún trabajo. El trabajo J se asigna a su procesamiento en el baño B' solamente si al tiempo T', J ya terminó de procesarse en el baño previo, B.

3) La regla 5.4.13 permite que el trabajo J' sea asignado al baño B, cuando de acuerdo a `finished(J,B,T),` el baño B ya terminó de procesar J, donde J ≠ J'. Además no hay razón para creer que J' ya fue procesado en B en un tiempo previo, T'', que sea menor o igual al tiempo T':

```
assigned(J,B,T,D) ← do(J,B,T,D), do(J,B',T',D'), job(J),        (5.4.11)
    T<T',D=T+D, B'==B+1, T' >=D, D <=deadline.
assigned(J,B',T',D') ← finished(J,B,T), do(J,B',T',D'),         (5.4.12)
    B'==B+1, T' >= T, F=T'+D', F<=deadline.
assigned(J',B,T',D') ← finished(J,B,T), do(J',B,T',D'),         (5.4.13)
    times(T''), not finished(J',B,T''),
    T' >= T, T'' <=T', F=T'+D', F<=deadline.
```

Adicionalmente, en la sección **DEFINE** se describe la regla para determinar, cuándo un trabajo debe considerarse como terminado, de acuerdo a lo definido en la función 5.10. Esto se consigue verificando si el trabajo J ya fue asignando al baño B en el tiempo T, o sea `assigned(J,B,T,D)`. Cuando esto ocurre, se asigna al tiempo de terminación, T', el resultado de la operación T + D, donde D es el tiempo de residencia de J en B. También se verifica que T' esté dentro del rango de fecha límite:

```
finished(J,B,T') ← assigned(J,B,T,D), T'=T+D, T < T',           (5.4.14)
    T' <= deadline.
```

Después de que los trabajos han sido asignados a los baños, hay que verificar que no haya traslape en el uso de recursos. Las funciones definidas en 5.11 y 5,12 son implementadas con las reglas de 5.4.15 a 5.4.20 y definen las situaciones en que ocurre este traslape. Estas reglas son bastante más complejas que las reglas anteriores y para implementarlas se hace uso de la negación por defecto, *not*,

el cual es una de las características más distintivas de ASP.

Primeramente, se verifica si un trabajo asignado a cierto baño no está traslapado con otro cierto trabajo dentro del mismo *answer set*. Las reglas 5.4.15 y 5.4.16 definen esta situación.

En la regla 5.4.15 que implementa la función definida en 5.11, se define cómo determinar si no hay traslape entre trabajos en el uso de un baño de agua. Así tenemos que un trabajo J' que ya terminó de ejecutarse en un baño B y es impar (B\2==0), o sea un baño químico. En consecuencia, no debe haber retrasos en la asignación al siguiente baño de agua. Entonces verificamos que cuando el trabajo J' sea asignado al siguiente baño B' en T', no haya otro trabajo J" asignado a B' con tiempo de inicio T". Sin embargo, no es suficiente con que se verifique si los trabajos J' y J" inician al mismo tiempo en el baño B'. También puede ocurrir que el trabajo J" haya iniciado antes que J' y aún no termine al tiempo que debería iniciar J'. Para determinar esta situación, entonces tenemos que verificar cual es el tiempo máximo en que terminarán todos los trabajos en el uso del baño B'. Después se debe contrastar este rango de tiempo con el período en que J' requiere ese baño. Esto se implementa con el agregado *#max* de Clingo, con el cual obtenemos el tiempo máximo que duraría ejecutándose cualquier J" en B'. El resultado se guarda en F y después se verifica si este valor es mayor que T. Si ese es el caso, significa que el baño B' está ocupado y por tanto la regla es verdadera:

```
busyBath(J',B,T,J',B',T',D',J”,T”,D”,F) ← finished(J',B,T),    (5.4.15)
    assigned(J',B',T',D'), job(J”), J” != J',
    assigned(J”,B',T”,D”), T'>T,  B\2 == 0, B'==B+1,
    F=#max{E:finished(J',B,T), assigned(J',B',T',D'),job(J”),
    J” !=J', assigned(J”,B',T”,D”),  E=T”+D”,
    finished(J”,B',E)}, T” <=T, F !=#inf, F==T',F>T.
```

Se definió también otra regla `busyBath` que tiene como único fin el de reducir el número de parámetros para facilitar su uso después:

```
busyBath(J2,B2,T2,D2) ←                                    (5.4.16)
    busyBath(J2,B1,T1,J2,B2,T2,D2,JT,T3,D3,F).
```

Aunque los trabajos pueden permanecer en espera en un baño de agua, en algunos modelos existe una espera injustificada en un baño de agua mientras el siguiente baño químico no está ocupado. Las

siguientes cuatro reglas identifican esta situación.

La regla `delayedBw(F),` implementa la función definida en 5.12, y define el caso en el cual un trabajo J termina de procesarse en el baño B (B es un baño de agua), al tiempo T, y hay un tiempo de espera innecesario entre la asignación de J al siguiente baño B'. Se dice que el retraso es innecesario porque en la regla se determina que no hay ningún trabajo utilizando ese baño B'. Este retraso se determina buscando cuáles son los tiempos de inicio de todos los trabajos en B', de entre esos tiempos se determina cual es el mínimo, mediante el agregado *#min*, que asigna el resultado de la comparación a F. Después, simplemente se verifica si F > T; si esto es verdadero, significa que ningún otro trabajo tiene asignado B' al tiempo T y por consiguiente, J está quedándose en espera en B sin razón alguna:

```
delayedBw(F) ← finished(J,B,T), F=#min{T' :                    (5.4.17)
    finished(J,B,T),assigned(J',B',T',D'), B'==B+1,job(J'),
    times(T'), bath(B') }, F !=#sup, F > T.
```

En la otra regla, `delayedBw(F),` se utiliza la negación por defecto (*default*) para resolver los casos en que resulta que el trabajo se considera como retrasado porque no tenemos evidencia suficiente de lo opuesto. Esta regla, llama a su vez a otra regla, `not neg_delayedBw(J',B',T',D').` Nótese el uso del operador `not` con el cual ASP implementa la negación por defecto. Intuitivamente, esta última expresión se interpreta como “no nay razón para creer que J' esté retrasado por causa justificada” (solo se justifica el retraso cuando el baño que se va a asignar está ocupado con otro proceso). Cuando la regla es verdadera, significa que no tenemos una causa para que J' esté retrasado, entonces, definitivamente se considera que J' está retrasado.

*NOTA: Ver regla* `neg_delayedBw(J',B',T',D')` *en 5.4.19*

```
delayedBw(J',B',T',D') ← finished(J',B,T),                     (5.4.18)
    assigned(J',B',T',D'), B'==B+1, B\2 == 0, T'>T,
    not neg_delayedBw(J',B',T',D').
```

Para diseñar las reglas que establecen cuándo un trabajo no está retrasado, fue necesario considerar que hay dos casos en los cuáles un trabajo no se considera que esté retrasado innecesariamente. El primero tiene que ver con el caso en que un trabajo J' que ya terminó de procesarse en el baño de agua B, no puede ser asignado al siguiente baño químico B' porque éste está ocupado por otro trabajo, como se muestra en la siguiente regla:

```
neg_delayedBw(J',B',T',D') ←                                    (5.4.19)
    busyBath(J',B,T,J',B',T',D',J",T",D",F).
```

Otra regla `neg_delayedBw`, nos devuelve que no tenemos evidencia de que el trabajo J' en el baño B' esté retrasado en su asignación, porque no hay evidencia que así lo demuestre, ya que si esta regla es verdadera es porque la sección "`not delayedBw(J',B',T',D')`" de dicha regla fue verdadera. Lo que significa que no tenemos evidencia de que el trabajo en cuestión esté retrasado:

```
neg_delayedBw(J',B',T',D') ← finished(J',B,T),                  (5.4.20)
    assigned(J',B',T',D'), B'==B+1, B\2 == 0, T'>T, job(J"),
    times(E), not delayedBw(J',B',T',D').
```

La última regla de la sección DEFINE, es la regla que define los casos en que no todos los trabajos logran terminar porque su tiempo de procesamiento excede el *deadline*. En este caso, necesitamos contar el número de trabajos terminados. Para hacer la cuenta, se usa el agregado *#count*.

```
jobsFin(N)← N= #count { J,T : job(J), finished(J,B,T) }.        (5.4.21)
```

• La parte PRUEBA del programa contiene las reglas sin cabeza que eliminan los *answer sets* que no son soluciones porque no cumplen con las restricciones del problema, y por lo tanto esos *answer sets* deben ser eliminados. Las reglas son:

```
← badStarting(J,B1,T1,D1,B2,T2,D2).                             (5.4.22)
← anyJobStartAtOne(T).                                          (5.4.23)
← badSincronyBch(T2).                                           (5.4.24)
← overlapBathJobs(T2).                                          (5.4.25)
← overlapJobBaths(T2).                                          (5.4.26)
← badSincInB1s(J,X,F1,J3,Z).                                    (5.4.27)
← delayedBw(F).                                                 (5.4.28)
← delayedBw(J2,B2,T2,D2).                                       (5.4.29)
← neg_delayedBw(J2,B2,T2,D2), not busyBath(J2,B2,T2,D2).        (5.4.30)
← numJobs(Nj), numBaths(Nb), jobsFin(N), N != Nj*Nb.            (5.4.31)
```

Sobre la restricción indicada en 5.4.31, es importante hacer notar que el número de trabajos y el número de baños no es conocido durante la realización de la codificación, puesto que es un parámetro que se agrega a tiempo de ejecución. Entonces, para decidir si es correcto el número de secuencias de

trabajo que nos devuelve `jobsFin(N)`, en esta regla tenemos que calcular cual es el dato exacto dados los parámetros *nb* y *nj*. Si el número no es correcto porque no todos los trabajos pudieron terminar, ese modelo no es una solución correcta y es eliminado.

Finalmente, para esta primera aproximación, describiremos cómo se implementó la optimización del plan, cuya función objetivo es $min[C_{max}]$ . En este caso, para hacer la optimización del *makespan* necesitamos conocer cual fue el trabajo con el tiempo más largo, de acuerdo a la definición dada en la función 5.13. Para conocer este dato, en la regla `lastJob(F)`, se usa el agregado *#max*, el cual nos devuelve el tiempo T de terminación más largo para cierto trabajo J procesándose en el último baño, Nb, y el resultado se asigna a F.

Después, en la regla 5.4.33 usamos la expresión de Clingo *#minimize*, la cual devuelve como resultado el valor mínimo del conjunto F, que es donde están los tiempos de terminación de todos los trabajos en el último baño. El valor mínimo de F devuelto por #minimize representa a $min[C_{max}]$

```
lastJob(F)← F=#max{T,J : finished(J,Nb,T),job(J),          (5.4.32)
            times(T),  numBaths(Nb)}.
#minimize {1,F: lastJob(F), finished(J,B,F) }.             (5.4.33)
```

### 5.4.3 Codificación Basada en Rangos

Una vez codificadas las restricciones del problema en ASP según se detalló en párrafos anteriores, resultó evidente que la máquina de inferencias era incapaz de resolver instancias de este problema con un número de trabajos y un número de baños mayor a 4. El resultado no es sorprendente si tomamos en cuenta que el problema en cuestión pertenece a la clase NP-hard (Röck, 1984). Es bien conocido que en esta clase de problemas, conforme crece la entrada, el número de combinaciones llega a ser tan enorme que el problema se torna intratable.

De hecho, para este problema en particular se encontró, en base a resultados experimentales, que el número de combinaciones N depende del número de baños, número de trabajos y fecha límite, de acuerdo a la siguiente ecuación:

$$N = D_d^{Nb*Nj}$$

donde *Nj* es el número de trabajos, *Nb* es el número de baños y $D_d$ es la fecha límite.

Para evidenciar el crecimiento exponencial de la función, en la tabla 5.4 se muestra cómo varía el número de combinaciones según varían los distintos parámetros de esta ecuación exponencial.

| Baños | Trabajos | Fecha limite | Relación | No. Modelos |
|---|---|---|---|---|
| 2 | 2 | 18 | $18^{2*2}$ | 104,976 |
| 3 | 2 | 25 | $25^{2*3}$ | 244,140,628 |
| 3 | 3 | 34 | $34^{3*3}$ | 60,717,000,000,000 |
| 4 | 4 | 50 | $50^{4*4}$ | 1.5259e+27 |
| 5 | 5 | 65 | $65^{5*5}$ | 2.1030e+45 |
| 6 | 6 | 83 | $83^{6*6}$ | 1.2213e+69 |
| 7 | 7 | 95 | $95^{7*7}$ | 8.0995e+96 |
| 8 | 8 | 120 | $120^{8*8}$ | 1.1684e+133 |

Tabla 5.1 Número de combinaciones que genera la ecuación N = $D_d^{Nb*Nj}$ conforme se varían sus parámetros.

Analizando la tabla, no es difícil comprender por qué resulta tan complejo resolver problemas de planificación y planificación de tareas (*scheduling*) con un enfoque combinatorio, como en ASP, puesto que resulta evidente que el factor de tiempo es determinante en el número de combinaciones resultantes. En este punto resulta pertinente aclarar que las máquinas de inferencia para solución de problemas representados en ASP, realizan la búsqueda de la solución mediante dos etapas: a) instanciación (*grounding*), y b) búsqueda de la solución. El proceso de instanciación consiste en reemplazar las variables de primer orden del programa ASP por instancias de base. Además, para reducir la explosión combinatoria, el proceso de instanciación se realiza utilizando la minimalidad de los modelos estables. Lamentablemente, esto es insuficiente para muchos problemas, y la etapa de instanciación, continúa siendo un cuello de botella en ASP, debido a que no maneja de manera eficiente dominios grandes de enteros y tampoco maneja dominios de números reales (De Cat et al., 2012), (Aziz, 2015).

El cuello de botella en la etapa de instanciación, es especialmente insalvable en problemas donde el modelo involucra tiempo, como por ejemplo en el problema de planificación de tareas abordado en esta investigación, ya que es evidente a partir de la tabla 5.5 que el dominio del problema es enorme, aún para instancias del problema con relativamente pocos trabajos y pocos baños.

Entonces, buscando aligerar el proceso de instanciación, se decidió reducir el tamaño del conjunto de posibles soluciones incidiendo directamente en la regla que genera estas secuencias. Se debe ser extremadamente cuidadoso en seleccionar las heurísticas que se aplicarán para reducir el número de combinaciones, puesto que es posible que inadvertidamente se eliminen conjuntos de

solución donde se encuentra la solución óptima.

La selección de las heurísticas se realizó basándose en conocimiento acerca del problema y sus soluciones. En particular, la heurística que se implementó se basa en generar combinaciones de posibles soluciones mediante rangos. Es decir, como el problema en estudio es sobre un taller de flujo, en el cual los trabajos deben procesarse obligadamente en orden secuencial e iniciando todos los trabajos en el primer recurso, entonces resulta irrelevante que se generen secuencias donde los trabajos inicien el procesamiento en el primer y segundo baño después de la mitad de tiempo disponible para terminar toda la secuencia de procesamiento, o viceversa. Ahora bien, como no sabemos de antemano cual de los trabajos se ejecuta primero, entonces se decidió hacer un modelado formal del proceso mediante el cual se realiza el cálculo de los rangos para generar las secuencias de posibles soluciones. Los cálculos se hicieron en tres etapas, y de acuerdo a las funciones definidas en 5.13 a la 5.17 relativas a la generación de combinaciones mediante rangos.

La siguiente regla implementa la función 5.14, y con ella calculamos para cada trabajo, la sumatoria de los tiempos de procesamiento, D, en todos los B baños, para cada trabajo J mediante el agregado *#sum* de Clingo. El resultado es asignado al conjunto L:

```
dBsumOneJ(J, L) ← job(W), J =1..W, L=#sum{D,B:  bath(M),      (5.4.34)
    B=1..M, job(W), J =1..W, duration(J,B,D)}.
```

La función 5.15 es implementada con la regla `maxDelayAnyJ(J,B,MaxR)` utilizando el resultado que nos devuelve la regla antes descrita, `dBsumOneJ(J,L).` Este resultado, `L`, se lo restamos a la fecha límite (*deadline*), y así obtenemos el máximo retraso, `MaxR`, permitido entre baños del trabajo `J`:

```
maxDelayAnyJ(J,B,MaxR)← J=1..W, job(W), bath(M),B=1..M,      (5.4.35)
    dBsumOneJ(J,L), MaxR=deadline-L.
```

Además de necesitar el máximo retraso posible del trabajo J, para calcular el rango en que se generan las combinaciones para cada baño, necesitamos la sumatoria de los tiempos de procesamiento hasta el baño en que se está calculando el rango, dada por la definición 5.16. Por ejemplo, si se está calculando el rango del trabajo J en el baño 5, necesitamos sumar las duraciones de procesamiento de ese trabajo en los baños 1..5. Para obtener este resultado, nuevamente usamos el agregado *#sum*:

```
dSumJuntilB(J,Buntil,L) ← job(J), bath(Buntil),              (5.4.36)
    L=#sum{D,B:  bath(Buntil), B=1..Buntil, duration(J,B,D)}.
```

Los resultados obtenidos en las reglas precedentes, se emplean para calcular el rango superior e inferior para cada baño del trabajo J. El cálculo se realiza siguiendo la función 5.17. El rango para las secuencias del baño B y trabajo J, está dado por el límite superior, `Ru`, y el límite inferior, `Rl.` El límite superior se calcula sumando el máximo retraso permitido para ese baño, más la sumatoria de los tiempos de procesamiento del trabajo `J` hasta el baño `B:`

```
rango(J,B,Ru,Rl) ← bath(B), job(J), duration(J,B,D),          (5.4.37)
    maxDelayAnyJ(J,B,MaxR), dSumJuntilB(J,B,L), Ru=MaxR+L,
    Rl=Ru-MaxR+1.
```

Por último, en la regla para generación de las secuencias `do(J,B,T,D)` se hace el llamado a la regla desde dónde se inicia el cálculo del rango, `rango(J,B,Ru,Rl)`, para los tiempos de inicio, `T`, para cada trabajo `J` y baño `B`, que deben quedarse dentro de los límites establecidos por las variables `Ru` y `Rl` de acuerdo a la ecuación 5.18:

```
{do(J,B,T,D) ← times(L), T=1..L, T+D<=deadline,          (5.4.38)
    rango(J,B,Ru,Rl), T>=Rl, T<=Ru }=1:- duration(J,B,D),
    job(W), J=1..W, bath(M), B=1..M.
```

## 5.5 Conclusiones del Capítulo

En este capítulo describió como se realizó el modelado del problema de planificación de tareas para un AWS de un Sistema de Manufactura de Semiconductores y cómo se realizó la implementación del modelo con el lenguaje de representación ASP. La principal dificultad que se tuvo en la etapa de implementación es debido al cuello de botella que representa la etapa de instanciación (*grounding*) en ASP. Si bien se usaron diferentes estrategias para resolver el problema con un número más grande de trabajos y de baños, y se logró un éxito relativo puesto que fue posible que ASP resolviese problemas de mayor tamaño que con la estrategia inicial, sin embargo, se detectó que este incremento no fue suficiente para resolver todas las instancias requeridas para hacer la comparación con los resultados de otros autores. La limitante de ASP proviene del enfoque que las máquinas de inferencia de ASP utilizan para solucionar el problema. Específicamente, es la primer etapa la causante del problema, ya que en esta etapa se instancían primero todas las variables y después se procede a la etapa de búsqueda. Este enfoque resulta impráctico para problemas en los cuáles la relación entre la entrada y el número

de combinaciones del modelo, crece de forma extraordinaria, como ocurre en la mayoría de los problemas de planificación.

# Capítulo 6

## RESULTADOS COMPUTACIONALES

En esta sección se presentan los resultados obtenidos para el problema de Optimización y Planificación de tareas de un Sistema Automatizado de Grabado (*Automated Wet-etch System*, AWS) de una empresa de semiconductores. Aunque en la literatura existen muchos casos de estudio relacionados con la planificación de tareas en Sistemas de Semiconductores, son pocas las que se enfocan directamente hacia AWS. No obstante que en la solución de este problema se han utilizado diversos enfoques, después de hacer una búsqueda bibliográfica extensa no encontramos evidencia de que la solución del problema se haya abordada utilizando teorías y técnicas basada en lógica no monotónica o del sentido común como las que se utilizaron en esta investigación. En particular, la implementación y solución del modelo matemático del problema fue resuelto utilizando el lenguaje ASP.

Uno de los principales obstáculos para la realización de los experimentos, es precisamente el acceso a una base de datos que corresponda a una empresa real. Después de realizar una búsqueda infructuosa para la localización de datos reales, se procedió a utilizar una serie de datos que fueron creados de manera artificial y publicados por Bhushan y Karimi (2003).

En la Tabla 6.2 aparecen los tiempos de procesamiento para 25 trabajos y 12 baños publicados inicialmente por dichos autores y utilizados en esta investigación. Estos datos han sido usados también por otros investigadores que plantearon una solución a la planificación de un AWS, como por ejemplo Zeballos y otros (2011). Sin embargo, la utilización directa de estos datos en ASP resultó imposible, puesto que tienen valores que pertenecen al conjunto de los números reales y ASP trabaja única y exclusivamente con números enteros. Entonces, se decidió redondear los tiempos de procesamiento a valores enteros.

Aunque el redondeo de los tiempos de procesamiento era necesario para realizar el mismo experimento que Zeballos y otros (2011), esto no era suficiente, también se requería que se redondearan los tiempos de transferencia del robot. En este caso no era suficiente con hacer un redondeo, puesto que la duración y diferencia entre los distintos tiempos de transferencia es del orden de milisegundos (Ver tabla 6.1).

| **τ1** | **τ2** | **τ3** | **τ4** | **τ5** | **τ6** | **τ7** | **τ8** | **τ9** | **τ10** | **τ11** | **τ12** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.2 | 0.6 | 0.8 | 1.0 | 0.4 | 0.6 | 1.0 | 1.0 | 0.8 | 0.4 | 0.8 | 1.0 |

Tabla 6.1 Tiempos de transferencia para el robot en un sistema AWS

Como puede apreciarse en la tabla 6.1, si por ejemplo, los tiempos en milisegundos, $\boldsymbol{\tau}_1$=1.2, $\boldsymbol{\tau}_2$=0.6, $\boldsymbol{\tau}_3$=0.8 se redondean, todos quedarían en con un tiempo igual a 1. En cambio, los tiempos $\boldsymbol{\tau}_5$=0.4 y $\boldsymbol{\tau}_{10}$=0.4, al redondearse quedarían en cero, lo cual resulta inverosímil.

Ahora bien, otra forma de que podía haber lidiado con estos datos, es el de escalarlos, multiplicándolos por 10. El problema con esta solución, es que el escalamiento también debería de realizarse a los tiempos de procesamiento. Si se analizan los datos de las tablas 6.2 y 6.3, puede percatarse que el tiempo mayor corresponde al trabajo 3 en el baño 7, el cual es igual a 13.0. Además a

través de experimentos se determinó que el orden de magnitud del *makespan* es alrededor de 400 segundos. Esta es también la fecha límite que debe asignarse para resolver los problemas más grandes. Ahora bien, si multiplicamos los tiempos de procesamiento por 10, lógicamente la magnitud del *makespan* también se incrementará en una cantidad proporcional, llevando esta magnitud a un valor alrededor de 4000 segundos.

Esto supone un problema importante en ASP, porque el número de combinaciones posibles es exponencial, y la base de este exponente es precisamente el límite de tiempo para que todos los trabajos terminen su procesamiento. Claramente, este enfoque resulta inadecuado para resolver el problema en ASP, ya que la resolución de un problema en este lenguaje se realiza en dos etapas: la etapa de instanciación (generación de combinaciones) y la etapa de búsqueda de la solución. La primera etapa es el cuello de botella de ASP, puesto que entre mayor sea el número de combinaciones, menos factible resulta resolver el problema en cuestión.

Por las anteriores argumentaciones, se simplificó el problema, descartando los tiempos de transferencia del robot y considerando solamente los tiempos de procesamiento de los trabajos en los diferentes baños.

Los datos redondeados de la tabla 6.2 se utilizaron para la solución de los problemas P1-P6 y P9 (La tabla 6.7 describe la configuración de estos problemas). Los datos redondeados de la tabla 6.3, fueron utilizados para resolver el problema denominado P7.

| **trabajos** | **baños** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 1 | 4.3 | 6.7 | 11.3 | 6.3 | 2.5 | 6.9 | 8.1 | 7.5 | 4.2 | 7.1 | 3.9 | 6.8 |
| 2 | 5.8 | 6.7 | 8.2 | 6.5 | 4.9 | 6.5 | 12.8 | 6.8 | 10.4 | 6.7 | 11.8 | 6.7 |
| 3 | 10.6 | 6.7 | 2.6 | 6.4 | 2.7 | 7.3 | 13.0 | 6.6 | 11.4 | 6.8 | 9.2 | 6.6 |
| 4 | 2.7 | 6.9 | 6.9 | 7.6 | 3.5 | 7.4 | 3.9 | 6.6 | 7.2 | 6.7 | 3.9 | 6.8 |
| 5 | 4.1 | 6.7 | 11.0 | 6.8 | 7.4 | 6.2 | 3.1 | 6.3 | 3.7 | 6.2 | 9.4 | 6.9 |
| 6 | 3.7 | 6.9 | 2.5 | 6.4 | 6.5 | 6.6 | 2.5 | 6.6 | 2.6 | 6.5 | 2.7 | 6.3 |
| 7 | 10.5 | 6.7 | 3.7 | 6.6 | 11.9 | 6.6 | 2.6 | 6.2 | 6.9 | 6.5 | 3.9 | 6.8 |
| 8 | 3.9 | 6.8 | 6.6 | 6.4 | 3.3 | 6.9 | 3.4 | 6.4 | 11.3 | 6.7 | 5.8 | 7.5 |
| 9 | 2.5 | 7.5 | 1.4 | 7.6 | 6.6 | 6.8 | 11.0 | 6.9 | 12.9 | 6.5 | 5.2 | 7.8 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10.8 | 6.7 | 10.1 | 6.5 | 2.5 | 6.6 | 2.7 | 7.1 | 4.6 | 6.5 | 11.4 | 6.3 |
| 11 | 8.7 | 6.2 | 4.2 | 7.2 | 6.1 | 6.2 | 5.9 | 6.5 | 4.6 | 6.7 | 8.8 | 6.6 |
| 12 | 7.0 | 6.3 | 7.2 | 6.6 | 2.7 | 6.7 | 8.9 | 7.1 | 2.9 | 6.7 | 6.4 | 6.8 |
| 13 | 9.1 | 6.8 | 2.8 | 6.4 | 5.9 | 6.4 | 5.9 | 6.9 | 10.4 | 6.9 | 8.8 | 6.5 |
| 14 | 2.7 | 6.1 | 11.4 | 6.9 | 7.7 | 6.4 | 5.1 | 6.2 | 4.7 | 6.9 | 10.0 | 6.8 |
| 15 | 2.8 | 6.8 | 6.8 | 6.3 | 4.2 | 6.7 | 8.5 | 6.6 | 5.7 | 6.5 | 4.3 | 6.9 |
| 16 | 5.7 | 6.9 | 2.8 | 7.1 | 4.7 | 6.1 | 3.9 | 6.9 | 4.4 | 6.4 | 2.7 | 6.3 |
| 17 | 2.5 | 7.6 | 6.7 | 6.5 | 2.6 | 6.4 | 3.4 | 7.2 | 2.9 | 6.7 | 7.8 | 6.4 |
| 18 | 3.9 | 6.8 | 12.1 | 6.8 | 2.7 | 6.3 | 9.3 | 6.2 | 4.7 | 6.3 | 2.6 | 6.8 |
| 19 | 9.7 | 6.7 | 7.6 | 6.4 | 10.9 | 6.9 | 2.6 | 6.7 | 4.6 | 6.6 | 10.1 | 6.3 |
| 20 | 2.6 | 6.7 | 2.9 | 6.5 | 10.4 | 6.9 | 2.6 | 6.7 | 11.5 | 6.6 | 3.7 | 6.2 |
| 21 | 4.7 | 6.6 | 4.9 | 6.9 | 2.6 | 6.8 | 12.7 | 6.2 | 2.6 | 6.7 | 6.9 | 6.4 |
| 22 | 2.5 | 6.3 | 2.6 | 6.6 | 7.9 | 6.8 | 12.5 | 6.8 | 2.6 | 6.5 | 7.8 | 6.4 |
| 23 | 11.4 | 6.4 | 8.9 | 6.6 | 2.7 | 6.4 | 11.4 | 7.4 | 11.3 | 6.8 | 2.9 | 6.9 |
| 24 | 6.8 | 6.5 | 2.8 | 7.5 | 3.9 | 7.2 | 9.8 | 6.5 | 8.6 | 6.3 | 11.8 | 6.2 |
| 25 | 8.8 | 6.9 | 8.8 | 6.8 | 11.3 | 6.8 | 11.3 | 6.1 | 6.7 | 6.5 | 2.6 | 6.4 |

Tabla 6.2 Tiempos de procesamiento para los trabajos P1-P6 y P9 (en segundos)

| **trabajos** | **baños** | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 11,1 | 6,68 | 5,24 | 6,92 |
| 2 | 8,47 | 6,35 | 10,1 | 7,02 |
| 3 | 9,19 | 6,35 | 4,6 | 6,71 |
| 4 | 10,8 | 7,12 | 10,2 | 6,83 |
| 5 | 7,4 | 7,05 | 4,07 | 6,58 |
| 6 | 10,8 | 6,76 | 1,01 | 6,37 |
| 7 | 3,48 | 6,67 | 1,41 | 6,46 |
| 8 | 2,51 | 6,23 | 8 | 6,23 |

Tabla 6.3 Tiempos de procesamiento para el trabajo P7 (en segundos)

## 6.1 Primera Versión del Programa ASP

Los primeros experimentos del problema consistieron en determinar la eficiencia de ASP para problemas de un orden menor, iniciando con 2 problemas y 2 baños, y luego sucesivamente ir incrementando tanto el número de trabajos como el de los baños.

La metodología empleada para el diseño de esta versión es la tradicional: Genera y Prueba. Es decir, primero se generan todas las combinaciones posibles. Después, mediante reglas diseñadas exprofeso, se eliminan las soluciones que no cumplen con las restricciones del problema.

En la tabla 6.3 se muestran los resultados obtenidos utilizando la primera versión del modelo descrito en el capítulo 8. La máquina de inferencias utilizada fue Clingo 4.5. Los resultados fueron obtenidos en una máquina a la que denominaremos M1, y sus características son: a) Procesador: Intel Core i5.3337 CPU @1.80GHz X4; b) Memoria: 5.6 Gi, y Sistema Operativo Ubuntu 15.04 de 64 bits.

Aunque la metodología de Genera y Prueba resulta muy conveniente para muchos problemas, también es conocido que la máquina de inferencias de ASP es muy eficiente para la solución de problemas difíciles, pero se ve seriamente limitada en la solución de problemas que tienen espacios de solución enormes, como el caso del problema de planificación de tareas. Esta limitación proviene sobre todo, no del módulo que busca la solución, sino por la estrategia empleada en el módulo de instanciación, la cual consiste en generar todas las posibles combinaciones del problema. Para problemas altamente dependientes del tiempo, como lo es la planificación de tareas, el tamaño del número de combinaciones tiene una función exponencial, en la cual la base del exponente es el tiempo que requiere el problema para procesarse, y por consecuencia, el tamaño del problema crece enormemente conforme este tiempo crece.

La tabla 6.4 contiene los resultados realizados para este primer experimento. En Clingo es posible especificar a tiempo de ejecución como se requiere hacer la búsqueda de la solución: a) para el caso si quiere buscar una sola solución es - *n* 1; b) para búsqueda de solución óptima es - *n* 0 e incluir la regla con el agregado `#minimize`, y c) la búsqueda de todas las soluciones se requiere con el comando - *n* 0, sin incluir el agregado de `#minimize`. Estas combinaciones de comandos se expresan en la tabla de la siguiente forma en la columna de comando de ejecución: 1) Los datos de la columna

denominada - *n* 1, se tienen resultados para la primer solución; 2) en la columna denominada - *n* 0, se tienen resultados para el caso en que se pidió buscar todas las soluciones, y 3) Por último en la columna etiquetada con - *n* 0 Opt, se tienen resultados para cuando se pidió buscar la solución óptima. Tanto en la tabla 6.4 como en las tablas 6.5 y 6.6, si el problema fué satisfacible, se indica con la abreviatura Sat.

| **Baños** | **Trabajos** | **Fecha límite** | **Modelos** | **Comando de Ejecución** | | | **Tiempo** | **CPU-tiempo** | **Makespan** |
|---|---|---|---|---|---|---|---|---|---|
| | | | | **-n 1** | **-n 0** | **-n 0 Opt** | | | |
| 2 | 2 | 18 | 1 | | | Sí | 1.820s | 0.120s | 18 |
| 2 | 2 | 18 | 6 | | Sat | | 0.177s | 0.170s | 14 |
| 2 | 2 | 18 | 1 | | | Sí | 0.172s | 0.170s | 14 |
| 2 | 2 | 18 | 6 | Sat. | | | 0.158s | 0.150s | 14 |
| 3 | 3 | 34 | 1 | | | Sí | 10.190s | 10.180s | 30 |
| 3 | 3 | 34 | 1 | | | Sí | 15.302s | 15.290s | 30 |
| 3 | 3 | 34 | 1 | Sat. | | | 8.132s | 8.060s | 30 |
| 4 | 4 | 50 | 1 | | | Sí | 96.959s | 96.920s | 50 |
| 4 | 4 | 60 | 1 | | | Sí | 967.335s | 418.100s | 59 |

Tabla 6.4 Resultados computacionales obtenidos en M1, usando ASP-Clingo, 1ra. Versión del Programa

Como puede inferirse, a partir de los resultados de la tabla 6.4, conforme crece el tamaño del problema, el tiempo de la solución también crece enormemente. Para problemas mayores a 4 trabajos y 4 baños no se encontró solución durante un tiempo menor a 3600 segundos.

## 6.2 Segunda Versión del Programa ASP

En vista de que con la primera versión del programa ASP, el tamaño máximo de la instancia del problema resuelto fue demasiado pequeño, se buscó como mejorar el programa utilizando los conceptos detallados en el capítulo 8. La mejora consistió en explotar el conocimiento del problema para reducir el número de combinaciones posible. Es decir, como el problema es secuencial, resulta innecesario generar combinaciones de posibles soluciones en cada baño para todo el rango de tiempo.

Entonces, en el programa se calculan los rangos en que se deben generar las combinaciones para cada baño. El rango resultante depende de cual trabajo sea y cuáles sean sus tiempos de procesamiento.

En la tabla 6.5 se muestran los resultados obtenidos con esta nueva versión del programa.

| Baños | Trabajos | Fecha límite | Modelos | Comando de Ejecución | | | Tiempo | CPU-tiempo | Makespan |
|---|---|---|---|---|---|---|---|---|---|
| | | | | -n 1 | -n 0 | -n 0 Opt | | | |
| 2 | 2 | 18 | 1 | | | Sí | 0.060s | 0.020s | 18 |
| 2 | 2 | 18 | 1 | Sat | | | 0.029s | 0.030s | 18 |
| 2 | 2 | 18 | 1 | | Sat | | 0.029s | 0.030s | 18 |
| 2 | 2 | 20 | 4 | | Sat | | 0.032s | 0.030s | 18 |
| 3 | 3 | 34 | 1 | | Sat | Sí | 0.557s | 0.550s | 33 |
| 4 | 4 | 50 | 1 | | | Sí | 11.258s | 11.250s | 50 |
| 4 | 4 | 50 | 36 | | Sat | | 15.302s | 15.290s | 50 |
| 5 | 5 | 65 | 2567 | | Sat | | 260.639s | 260.500s | 64 |
| 5 | 5 | 65 | 1 | | | Sí | 167.941s | 165.890s | 65 |

Tabla 6.5 Resultados computacionales obtenidos en M1, usando ASP-Clingo, 2da. Versión del Programa

Como puede apreciarse a simple vista, aunque el valor del *makespan* es igual en la mayoría de los casos, se reducen notablemente los tiempos de solución para todas las instancias, e inclusive, es posible resolver problemas de mayor tamaño que con la versión anterior. Concretamente, es posible resolver el problema que consta de 5 trabajos y 5 baños.

Para determinar experimentalmente si la capacidad de cómputo impacta determinantemente en la solución del tamaño del problema, se realizaron experimentos con otra máquina que tiene casi el doble de capacidad del equipo empleado anteriormente. Más precisamente, las características de esta máquina, a la que de aquí en delante denominaremos M2, son: CPU @2.88GHz con 8 núcleos, Memoria de 16 Gigabytes, Sistema Operativo Linux de 64 bits. En este equipo se pudo resolver el problema con 6 trabajos y 6 baños. Para el problema de 7 trabajos y 7 baños no hubo solución. En este sistema no fue posible resolver problemas con 8 ó más trabajos y 8 ó más máquinas. La tabla 6.6 muestra los resultados obtenidas para problemas con 6 trabajos y 6 baños.

| baños | trabajos | Fecha límite | Modelos | Comando de Ejecución | | | Tiempo | CPU-tiempo | Makespan |
|---|---|---|---|---|---|---|---|---|---|
| | | | | -n 1 | -n 0 | -n 0 Opt | | | |
| 6 | 6 | 83 | 1 | Sat | | | 181.104s | 181.170s | 83 |
| 6 | 6 | 83 | 1 | | | Sí | 240.216s | 240.280s | 83 |
| 7 | 7 | 95 | 0 | | | No | 45799.1 | 45805.990s | Na |

Tabla 6.6 Resultados computacionales obtenidos en M2, usando ASP-Clingo, 2da. Versión del Programa

Las modificaciones realizadas al programa ASP y los experimentos realizados en equipos con capacidad diferente, solo confirmaron la teoría que predice que un sistema combinatorio es intratable conforme la entrada crece y que no influye notablemente el uso de un hardware más poderoso. En este caso, con un equipo que es aproximadamente el doble de capacidad de cómputo y más del doble de memoria RAM que la del primer equipo, solo fue posible incrementar el orden del problema resuelto en 2 niveles más. Es decir, se pasó de 5j x 5b a 7j x 7b. Aunque la solución para el problema 7j x 7b, también fue encontrada, ésta resultó ser insatisfacible.

Expresado de forma concisa, este nuevo enfoque, reduce la generación de posibles soluciones desde el rango completo de tiempo (1..*deadline*) a rangos particulares para cada trabajo y cada baño. A continuación se muestran algunas de las secuencias generadas por el enfoque sin rangos. Los parámetros de la combinación son `do(J,B,T,D)`, donde J es el número de trabajo, B es el número de baño, T es el tiempo de inicio del trabajo y D es el tiempo de procesamiento de J en B:

a) `do(1,1,7,4) do(1,2,4,6) do(2,1,2,6) do(2,2,11,7),`

b) `do(1,1,7,4) do(1,2,4,6) do(2,1,2,6) do(2,2,8,7),`

c) `do(1,1,1,4) do(1,2,16,6) do(2,1,18,6) do(2,2,6,7),`

d) `do(1,1,1,4) do(1,2,10,6) do(2,1,18,6) do(2,2,13,7).`

Como puede apreciarse en estas secuencias, existen combinaciones que representan solo una carga computacional. Por ejemplo, para la secuencia indicada en a), `do(1,1,7,4)`, con J=1, en B=1, terminará hasta T=11. Entonces es totalmente intrascendente que se generen secuencias como `do(1,2,4,6)` para J=1 en B=2 y T=4, pues a este tiempo, J=1 aún no ha terminado de procesarse en B=1.

En el otro extremo, tenemos el caso de la secuencia indicada en el inciso d). Por ejemplo, en `do(2,1,18,6)` y `do(2,2,13,7)`, tenemos que J=2 en B=1, tiempo de inicio T=18, y el mismo trabajo, J=2 inicia en B=2 a T=13. Estas secuencias rompen completamente con las restricciones de secuenciación del problema, puesto que necesariamente, todos los trabajos deben ser procesados primero en B=1 y después en B=2, y así sucesivamente.

En cambio, para el enfoque por rangos, se consideran tanto los aspectos de secuenciación como la fecha límite en que deben ser asignados todos los trabajos a todos los baños. A continuación se muestran los rangos calculados para el mismo problema de 2 trabajos y 2 baños. Los parámetros de rango son: `rango(J,B,T,D)`. J representa el trabajo, B el baño, T es el límite superior del rango y representa también el tiempo más tarde en que pudiese ser asignado un dado trabajo J a cierto baño B y que aún pudiese ser posible que el trabajo termine en tiempo. Así tenemos que por ejemplo para el `rango(1,2,12,5)`, con J=1 y B=2, el trabajo debe ser asignado a dicho baño cuando muy tarde a T=12. Sí así ocurre, el tiempo de terminación F de J=1, B=2, sería F=17. Para J=2 y B=2, al rango superior sería T=11, con lo cual terminaría en F=18:

```
rango(1,1,8,1) rango(1,2,12,5) rango(2,1,5,1) rango(2,2,11,7)
```

Algunos ejemplos de secuencias combinatorias generados dentro de estos rangos son las siguientes:

a) `do(1,1,2,4)  do(1,2,9,6)  do(2,1,1,6)  do(2,2,11,7)`

b) `do(1,1,1,4)  do(1,2,12,6)  do(2,1,1,6)  do(2,2,10,7)`

c) `do(1,1,5,4)  do(1,2,5,6)  do(2,1,5,6)  do(2,2,7,7)`

b) `do(1,1,8,4)  do(1,2,11,6)  do(2,1,5,6)  do(2,2,7,7)`

Como puede comprobarse fácilmente, todas estas secuencias fueron debidamente generadas para los rangos establecidos previamente.

Aunque esta mejora en el modelo solo nos permitió incrementar el tamaño del problema hasta J=7 y B=7, no por eso las reducciones en el número de combinaciones son importantes. Por ejemplo, para el problema de 2 trabajos y 2 baños, el número de combinaciones obtenidas con la versión 1 del programa, operando sobre el rango completo fue de 104,976, sin incluir la restricción de límite de tiempo. Con el enfoque de generación de combinaciones basada en rangos, el número de combinaciones se redujo a 1,600. Es decir, el número de combinaciones obtenido mediante rangos representa un 1.5242% respecto del tamaño del rango completo. Aún más la reducción del espacio permite conservar las soluciones óptimas que se encuentran dentro del rango completo.

### 6.3 Resultados con enfoque de solución ASP+CSP

Después de realizados los anteriores experimentos, resultó evidente que los tiempos de solución no mejorarían notablemente si simplemente se busca reducir el porcentaje de secuencias combinatorias y/o se usa un sistema de cómputo más potente. Lo que realmente se necesita para resolver problemas con secuencias combinatorias masivas, es un nuevo enfoque acerca de la etapa de instanciación del problema.

En la búsqueda en la literatura de este nuevo enfoque, se encontró que se han propuesto varias teorías y metodologías para sobrepasar el problema de instanciación en Clingo. En este caso, requeríamos no sólo de la teoría más relevante, sino también de que existiese libremente disponible el motor de inferencias que implementa la teoría y además que soportase el lenguaje ASP.

Así, se decidió por la máquina de inferencias híbrida Clingcon, el cual combina el lenguaje de modelado y representación del conocimiento, ASP, y a su vez soporta el uso de restricciones no-booleanas del área de Programación con Restricciones (Ostrowski & Schaub, 2012; Ostrowski, 2012). Clingcon sigue el enfoque de los modernos motores de inferencia. Este enfoque se conoce con el nombre de Teorías de Módulos de Satisfiabilidad (*Satisfiability Modulo Theories, SMT*) (Biere et al. 2009). Clingcon 2.0.3 se utiliza en la etapa de instanciación al motor de inferencias geocode 3.7.1 y en la etapa de búsqueda de la solución a Clingo 3.0.4.

Las capacidades extendidas de Clingcon permiten operar con variables globales de restricciones y expresiones de optimización sobre las variables de restricción. La ventaja más destacada de Clingcon es que soporta técnicas de aprendizaje sofisticadas para la interacción entre ASP y el motor de inferencias de CP.

El programa de planificación de tareas para un AWS de semiconductores para Clingcon implementa las funciones 5.4.5 a 5.4.10 descritas en el capítulo 5.

### 5.3.1 Diseño del experimento con el programa ASP+CSP

Con fines comparativos, este experimento se planteó de manera similar a Zeballos y otros (2011), con solo pequeñas diferencias. En concreto, se resolvieron los problemas identificados en la tabla 6.7 con la numeración que va de P1 a P6 y P9 utilizando los datos mostrados en la tabla 6.2. El problema P7 se resolvió con los datos de la tabla 6.4. El problema identificado por Zeballos como P8, no se resolvió porque éste es solo una modificación de los tiempos de transferencia del robot relativo al problema P7.

Como no estamos considerando los tiempos de transferencia del robot para ninguno de los trabajos, en este caso resulta inaplicable realizar este ajuste

| **Id. del Problema** | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** | **P9** |
|---|---|---|---|---|---|---|---|---|---|
| **Baños** | 6 | 6 | 6 | 12 | 12 | 12 | 4 | - | 12 |
| **Trabajos** | 5 | 15 | 25 | 5 | 15 | 25 | 8 | - | 10 |

Tabla 6.7 Identificadores de Problemas Resueltos

### 6.3.2 Resultados obtenidos en ASP+CSP con Clingcon

En la tabla 6.8 se muestran los resultados obtenidos para el problema de planificación de tareas AWS, tanto para la primer solución como para la solución óptima. Es importante aclarar que en esta investigación no se consideraron los tiempos de transferencia del robot.

| **Id Prob.** | **Primer Solución** | | | **Solución Óptima** | |
|---|---|---|---|---|---|
| | **makespan** | **tiempoCPU -** | | **makespan** | **Tiempo -CPU** |
| P1 [6 x 5] | 84 | 0.010s | | 71 | 506 (a) |
| P2 [6 x 15] | 177 | 0,100s | | 174 (NoOpt) | 30302750s |
| P3 [6 x 25] | 284 | 0,230s | | NS | NS |
| P4 [12 x 5] | 158 | 0,020 s | | 133 | 1440,68 a |
| P5 [12 x 15] | 253 | 0.180 s | | 361 | 3600 a |
| P6 [12 x 25 ] | 370 | 0.550s | | | |

Tabla 6.8 Resultados obtenidos para el problema AWS con ASP+CSP y Clingcon.

a =Imposible probar optimalidad entro de un tiempo=3600 seg.
NS = No solución encontrada

Al revisar los resultados obtenidos con Clingcon para el programa en ASP+CSP, resulta evidente que es mucho más conveniente utilizar una metodología híbrida, ASP+CSP, para la solución de problemas de datos masivos. Aunque el enfoque ASP+CSP que provee Clingcon, no resuelve completamente el problema de cuello de botella que tiene ASP en la etapa de instanciación, si sobrepasa a éste por dos órdenes de magnitud (Gebser et al., 2009). Por ello, es posible resolver en Clingcon muchos problemas, en los cuales ASP no encuentra siquiera una única solución.

Desafortunadamente, puede observarse también que para la mayoría de los problemas planteados tampoco se encontró la solución óptima con Clingcon.

### 6.3.3 Comparación de resultados obtenidos en MILP, CP+GVDR y ASP+CSP

Aunque el área de investigación de Razonamiento No Monotónico y ASP es una área muy activa y en constante expansión, existen pocas aplicaciones de ASP en la solución de planificación de tareas aplicadas al área de sistemas de manufactura. Entonces, es importante comparar los resultados obtenidos en esta investigación con resultados obtenidos con otros métodos para el mismo problema de planificación de tareas de un AWS de una fábrica de Semiconductores. Por ello, los resultados obtenidos con Clingcon se comparan con los resultados publicados por Zeballos y otros (2011), quiénes a su vez, compararon sus resultados obtenidos con el método CP+GVDR con los resultados obtenidos anteriormente por Bhushan & Karimi (2003) utilizando MILP.

Aunque como ya se aclaró en los párrafos anteriores, en esta investigación no se hizo la planificación de tareas considerando los tiempos del robot, estimamos que resulte pertinente esta comparación con los resultados obtenidos por Zeballos (2011) y Bhushan & Karimi (2003). La comparación se justifica porque los tiempos de transferencia del robot son mucho más pequeños que los tiempos de procesamiento de los trabajos en los baños. Inclusive el tiempo mayor de transferencia del robot es de 1.2 segundos y el menor de 0.4, de forma que el tiempo promedio de transferencia del robot es de 0.96. Entonces, en promedio los tiempos de inicio para cada trabajo y en cada baño podrían diferir en .96 segundos.

| Id Prob. | Primer Solución | | Solución Óptima | | Enfoque |
|---|---|---|---|---|---|
| | makespan | Tiempo CPU | makespan | Tiempo CPU | |
| P1 [6 x 5] | 218.1 | 0.01 | 82.6 | 0.94 | MILP |
| | 92.6 | 0.01 | 82.6 | 2,84 | CP+GVDR |
| | 84 | 0.010s | 70 | 1228.36 a | ASP+CSP |
| P2 [6 x 15] | 196.1 | 1687 | 195,2 | 3600 a | MILP |
| | 205.4 | 0,14 | 185 | 350 a | CP+GVDR |
| | 177 | 0,100s | 174 | 3600 a | ASP+CSP |

| | | | | | |
|---|---|---|---|---|---|
| P3 [6 x 25] | NS | - | NS | 3600 a | MILP |
| | 325,1 | 0,53 | 297,3 | 1346 a | CP+GVDR |
| | 284 | 0,230s | NS | NS | ASP+CSP |
| P4 [12 x 5] | 154.4 | 2,38 | 144.1 | (7.39)14.49 | MILP |
| | 161.5 | 0,06 | 144.1 | 0.39 a | CP+GVDR |
| | 158 | 0,020 s | 133 | 1440,68 a | ASP+CSP |
| P5 [12 x 15] | NS | - | NS | 3600 a | MILP |
| | 294.0 | 0.76 | 273.2 | 949 s | CP+GVDR |
| | 262 | 0.170 s | 246 | 0,180 s a | ASP+CSP |
| P6 [12 x 25 ] | NS | - | NS | 3600 a | MILP |
| | 497.5 | 17.29 | 443.4 | 493.37 | CP+GVDR |
| | 370 | 0.550s | N 361 | 3600 a | ASP+CSP |
| P7 [4 x 8 ] | 139.1 | 3.45 | 120.47 | (72.34)152 | MILP |
| | 128.20 | 0.05 | 120.47 | 1.40 a | CP+GVDR |
| | 103 | 0.04 | 91 | 3600 a | ASP+CSP |
| P9 [12 x 10] | 206.30 | 3452 | 206.30 | 3452 a | MILP |
| | 232.8 | 0.38 | 199.0 | 3440 a | CP+GVDR |
| | 207 | 0.08 | 193 | 3600 a | ASP+CSP |

Tabla 6.9 Comparación de resultados para el problema AWS con MILP, CP+GVDR y AWS+CSP (tiempos en segundos)

a =Imposible probar optimalidad entro de un tiempo=3600 seg.
NS = No solución encontrada

## 6.4 Discusión de Resultados

Aunque ASP es un paradigma de representación del conocimiento y razonamiento muy adecuado para la solución de problemas complejos, experimentalmente quedó demostrado que en la solución de problemas combinatorios con espacios de búsqueda masiva, como en el problema motivo de esta investigación, resulta evidente que su poder resolutivo se ve seriamente mermado. Aunque el problema de estudio es NP-difícil, y por consiguiente intratable en la búsqueda de la solución óptima, es posible que siguiendo un enfoque diferente para la generación del espacio de búsqueda se encuentren soluciones para problemas más grandes que con el enfoque actual de ASP.

Como el problema de cuello de botella de ASP es la generación de combinaciones, usando un enfoque híbrido del tipo ASP+CSP es posible sobrellevar en parte la sobrecarga impuesta por la etapa de instanciación del motor de inferencias, tal y como se demuestra con los resultados obtenidos en este

experimento.

Ahora bien, examinando los resultados obtenidos mediante el enfoque de ASP+CSP usado en esta investigación, es posible percatarse que ASP+CSP es claramente superior en cuanto al tiempo que se tarda en obtener la primer solución, la cual en todos los casos es superior tanto a MILP como a CP+GVDR, excepto en el problema P1, donde ASP+CSP obtiene el mismo tiempo que con CP+GVDR. Por cuanto hace al *makespan*, también con ASP+CSP se obtienen los valores menores que con los otros paradigmas, excepto para el problema P4, donde MILP obtiene un *makespan* de 154.4 y ASP+CSP de 158.

Por cuanto hace a las soluciones óptimas, si bien con ASP+CSP no fue posible encontrar ninguna solución óptima para ninguno de los problemas planteados en un tiempo menor o igual a 3600 segundos. Sin embargo, sí se encontraron otras soluciones no óptimas en las cuales los valores del *makespan* obtenido con ASP+CSP fueron menores para todos los problemas que los obtenidos mediante CP+GVDR y MILP.

# Capítulo 7

# CONCLUSIONES

En esta disertación se propone la solución del problema de planificación de tareas y la optimización del plan para un AWS en fábricas de semiconductores con un enfoque basado en razonamiento no monotónico. En particular se propone un enfoque híbrido, basado en la extensión de ASP con restricciones. En esta investigación se discute también porqué es importante explorar el diseño de planificadores con métodos basados en razonamiento del sentido común.

El desarrollo de esta tesis se realizó en tres etapas: En la primer etapa se realizó el modelado e implementación del problema siguiendo el enfoque convencional en el área de ASP, es decir mediante

la metodología Genera y Prueba. Durante la fase Genera, se diseño una regla para la generación de todas las posibles combinaciones de trabajos, baños y tiempos de inicio dentro del rango desde tiempo 1 hasta la fecha de *deadline*. Después de realizar varios experimentos, en los cuáles se asignó hasta un término máximo de 7200 segundos para la solución del problema, se detectó que Clingo es incapaz de resolver problemas de tamaño mayor a 5 trabajos y 5 baños. Experimentalmente se determinó que el número de combinaciones crece exponencialmente de acuerdo a la función $T^{Nb*Nj}$ , donde *T* es la fecha límite, *Nb* es el número de baños y *Nj* es el número de trabajos.

Este limitación de Clingo 4.5 para resolver el problema proviene del enfoque que se utiliza en el diseño de los motores de inferencias de ASP en el estado del arte, incluído Clingo 4.5, ya que realizan el procedimiento de razonamiento en dos etapas: primero genera las combinaciones, es decir instancía las variables con los datos del problema, y después, en la segunda etapa, busca los modelos o soluciones del problema. Esto deriva en la principal limitante de los resolvedores de ASP: requieren una cantidad de memoria inmensa para realizar la instanciación. Aunque este problema es mitigado en cierta forma mediante las técnicas inteligentes de instanciación que hacen una evaluación parcial de las reglas del programa y cuando es posible, eliminan instancias de la regla que no son aplicables en la solución del problema. Sin embargo, esta técnica no es lo suficientemente efectiva en algunas clases de programas en los cuales se tenga un problema cuya entrada es exponencialmente masiva, como en el caso de este problema de planificación de tareas y no es posible obtener soluciones en un tiempo que resulte de utilidad práctica.

En la segunda fase, buscando sobrepasar la limitante que supone el proceso de instanciacion en la máquina de inferencias Clingo, se propuso un método basado en rangos mediante el cual se busca reducir el número de combinaciones, explotando las características de ejecución del procesamiento del producto en el problema planteado. Es decir, puesto que el producto se procesa secuencialmente, resulta más conveniente que no se considere generar combinaciones con tiempo más allá de un cierto rango que depende de la etapa en que se procese el trabajo en ese baño. Este procedimiento nos permitió reducir el espacio de búsqueda hasta en un 98.5%. Esta reducción, aunque importante, solo nos permitió incrementar el tamaño del problema resuelto de 5 baños y 5 trabajos a 6 baños y 6 trabajos. Una vez que analizamos cual sería el número de combinaciones, no resulta tan sorprendente que no fuese posible mejorar sensiblemente el tamaño del problema resuelto por Clingo. Por ejemplo, para una fecha límite de 83, con 6 trabajos y 6 baños, el número de combinaciones es 8.0995e+96. Si el método por rangos nos reduce el número de combinaciones en aproximadamente un 98.5%, aún se

tiene un espacio de búsqueda igual a 1.2149e+95 combinaciones.

Evidentemente, era necesario considerar un nuevo enfoque del problema. Entonces, se procedió a analizar si de entre las numerosas extensiones y sistemas híbridos propuestos para trabajar con ASP, existía alguno que nos permitiese lidiar con el problema de instanciacion. Un análisis cuidadoso nos permitió detectar que se había propuesto recientemente una combinación de ASP con CSP para lidiar con problemas con datos masivos. De hecho, existen también varias propuestas de cómo hacer esta combinación de paradigmas: algunos proponen una integración bajo el concepto de Teorías de Módulos, otros proponen un enfoque basado en traducción en el cual todas las partes del modelo CSP son mapeadas a ASP. En este enfoque se seleccionó un sistema híbrido basado en Teorías de Módulo ya que los resolvedores disponibles están basadas en Clingo 4.5, el cual es hasta la fecha la máquina de inferencias más competente para ASP.

En la tercera etapa, se integró la generación de ASP y CSP para reducir los requerimientos de memoria, generando dos tipos de reglas: a) el conjunto de reglas regulares con predicados sobre dominios que pueden ser manejados por los resolvedores estándar de ASP, y b) el conjunto de reglas de restricciones sobre variables globales, en este caso fue la variable relacionada al tiempo. Este tipo de variables pueden ser manejados por los resolvedores de CSP sin instanciación (y por ello se pueden manejar aquí grandes dominios).

Los resultados experimentales demuestran que la resolución del problema mediante la utilización de un paradigma híbrido basado en ASP y CSP, mejora notablemente el rango de aplicabilidad de ASP, ya que fue posible encontrar al menos una primera solución para todos los problemas propuestos. Además, tanto el *makespan* como los tiempos de cómputo fueron mejores que los resultados obtenidos por otros investigadores con el paradigma CSP y MILP.

Sin embargo, aún bajo este paradigma no fue posible encontrar un resultado óptimo para todos los problemas propuestos, sino solo para los problemas P1 y P4. Sin embargo, tanto para los problemas para los cuales se encontró la solución óptima como para los que no se encontró ésta, el *makespan* mínimo obtenido por el paradigma ASP+CSP, fue consistentemente mejor que los otros dos paradigmas con los cuáles fueron comparados los resultados de ASP+CSP.

Estos resultados indican que el paradigma híbrido ASP+CSP se encuentra bien situado para la solución de problemas complejos y con un espacio de búsqueda que crece exponencialmente en relacion a la entrada.

La ventaja agregada con respecto a otros paradigmas, es que se puede extender fácilmente el modelo y el programa diseñado, para la solución de problemas con eventos estocásticos, como por ejemplo ruptura de máquina ó llegada de órdenes urgentes, aprovechando las características no monotónicas de ASP.

El paradigma seleccionado para la solución del problema, facilita además la integración del planificador de tareas con el planificador general en el que se incorpore información de otros niveles como ventas, almacén, cadena de proveedores, o cualquier otra información relacionada con el proceso de producción.

## Capítulo 8

## TRABAJO FUTURO

Existen muchas formas de extender esta investigación, tanto desde el punto de vista de las características del problema como desde el punto de vista de nuevas y diferentes metodologías y técnicas que pueden ser exploradas para la solución del problema.

Por lo que respecta al primer enfoque propuesto, el problema puede ser extendido para incluír uno o más robots para la transferencia de lotes entre baños. Aún más interesante sería, abordar el problema desde un punto de vista diferente, es decir, en lugar de hacerlo con un enfoque estático, abordarlo como un problema dinámico y estocástico y de esta forma, considerar la posible ocurrencia

de diversos eventos externos, tales como órdenes urgentes ó ruptura de máquina. También resultaría interesante que en lugar de buscar la optimización del plan, se buscara diseñar un plan robusto. La robustez del plan se puede obtener utilizando diversas medidas. La que resulta más atractiva es minimizar el nerviosismo del taller de piso. Por ejemplo, ante una ruptura de máquina, evitar diseñar el nuevo plan desde cero para evitar los costos de asignación y posible reconfiguracion de las máquinas. En su lugar, solo deben reasignarse los lotes que estén pendientes de procesarse y que dependan de la máquina que falló.

Ahora bien, para sobrepasar la limitación que tiene ASP con datos masivos, manejo de números reales y funciones, se pueden explorar algunas de las otras extensiones ó integraciones que han sido propuestas para ASP. En esta tesis se propone el uso del lenguaje ASPMT, el cual está basado en ASP, pero extiende este último para permitir fórmulas proposicionales multi-valuadas y fórmulas de primer orden. Aunque en ASP se pueden representar fluentes funcionales, éstos están basadados en predicados, y conforme la instancia crece, el dominio llega a ser excesivamente grande. Por otra parte los modelos ó *answer sets* de ASP no son otra cosa que los modelos de Herbrand, y por lo tanto no pueden soportar funciones en el sentido amplio. Además la no monotonicidad de ASP tiene que ver con la minimización de predicados y no tiene nada que ver con la minimización de funciones.

El lenguaje ASPMT no solo está basado en el paradigma ASP, sino también en el paradigma de Teorías Módulo de Satisfabilidad (*Satisfiability Modulo Theories, SMT*). SMT es un problema de decisión para fórmulas lógicas con respecto a otras teorías subyacentes que están expresadas en lógica clásica de primer orden. Algunas de estas teorías son lógica de diferencias, aritmética lineal sobre números racionales, aritmética no lineal sobre racionales, teoría de diversas estructuras de datos tales como listas, arreglos, y/o combinaciones de los mismos.

A principios del milenio, con el desarrollo de máquinas de inferencia SAT muy eficientes, surgió la feliz idea de conjuntar SAT y teorías de módulo y fueron creadas múltiples máquinas de inferencia SMT basadas en SAT, como por ejemplo: Ario, Barcelogic, CVC, CVC Lite, CVC3, ExtSAT, Harvey, HTP, ICS (SRI), Jat, MathSAT, Sateen, Simplify, STeP, STP, SVC, TSAT, UCLID, Yices (SRI), Zap (Microsoft), Z3 (Microsoft), iSAT.

Así tenemos que aún y cuando ASP es un exitoso paradigma de programación declarativa no monotonica, tiene limitaciones para el manejo de funciones. En tanto que, SMT es un paradigma que permite resolver algo de razonamiento especializado de primer orden, pero está limitado en el

razonamiento no monotónico, En resumidas cuentas, tanto ASP como SMT tienen fortalezas y debilidades, por lo que el diseño de un lenguaje como ASPMT es un paso lógico que busca integrar en un solo lenguaje lo mejor de los mundos ASP y SMT.

Aunque antes del desarrollo de ASPMT ya existía la integración de ASP con otros paradigmas, como por ejemplo la integración ASP+CSP que utilizamos en esta tesis para la resolución del problema propuesto, esta integración consiste en considerar como una caja negra las teorías con las que se extiende ASP. En cambio, ASPMT se basa en una nueva teoría conocida como semántica funcional del modelo estable (*Functional Stable Model Semantics, FSM*), la cual puede expresar valores por defecto asignados a las funciones. Adicionalmente, al estar ASPMT basado en FSM, esto permite realizar una integración fuerte entre ASP y otros lenguajes donde las funciones son las construcciones primitivas. No menos importante resulta saber que ya existen máquinas de inferencia para ASPMT que son competitivas con respecto a las mejores máquinas de inferencia para ASP. Estas máquinas son MVSM y ASPMT2SMT. Ambas utilizan un enfoque de múltiples herramientas, pero MVSM recae en el resolvedor Gringo/ClaspD, diseñado para ASP. En tanto, ASPMT2SMT utiliza como motor de inferencias a Z3, el cual es un resolvedor para SMT.

Otras características extendidas de ASPMT permiten resolver problemas que requieran el manejo de tiempo continuo. Por ejemplo, es posible modelar y desarrollar con ASPMT planes con tiempo continuo y hacer razonamiento sobre efectos acumulativos en cambios continuos, procesos y razonamiento espacial no monotónico tanto cualitativo como cuantitativo.

Además los lenguajes de acción, que son lenguajes de alto nivel diseñados para representar conocimiento acerca de acciones y cambio de forma concisa, también pueden ser reducidos a ASPMT y con ello facilitar el modelado de problemas donde se requiere hacer razonamiento acerca de acciones.

Por último, para el modelado del sistema completo de manufactura de semiconductores, se propone como trabajo futura una integración de ASPMT con redes de Petri, puesto que el proceso de producción de semiconductores requiere de reentrancia, por ello, es necesario utilizar un método de modelado que pueda manejar la competencias de recursos y evitar su bloqueo (*deadlock*). El uso de las redes de Petri para el modelado de sistemas de manufactura de semiconductores no es una propuesta novedosa, ya que se ha utilizado desde hace tiempo. En cambio, el modelado e implementación de la conjunción de los paradigmas redes de Petri y ASPMT sí lo es y permitirá asegurar que el modelado del sistema esté libre de bloqueos y asegurar que es correcto. Por otra parte la incorporación de ASPMT

permitirá generar planes para entornos cambiantes, como lo es en alto grado, los sistemas de manufactura en general y en particular el de los semiconductores.

**Referencias**


1. Abío, I.: Solving hard industrial combinatorial problems with SAT. Ph.D. thesis, Technical University of Catalonia (UPC) (2013).
2. Alviano, M., Faber, W., Gebser, M.: Rewriting recursive aggregates in answer set programming: back to monotonicit. Theory and Practice of Logic Programming (TPLP). Vol. 15, pp.559-573 (2015)
3. Antoniou, G.: Verification and correctness issues for nonmonotonic knowledge bases, International Journal of Intelligent Systems,Vol. 12, Issue 10, pp.725-738 (1997).
4. Anwar, S., Baral, C., Inoue, K.: Encoding higher level extensions of petri nets in answer set programming. In: LPNMR 2013, Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning, Corunna, Spain, pp.116-121 (Septembre 2013).

5. Aytug, H., Lawley M.A., McKay, K., Mohan, S. and Uzsoy, R.: Executing production schedules in the face of uncertainties: a review and some future directions, European Journal of Operational Research, Vol. 161, Issue 1, pp.86-110 (2005).
6. Azab, A. And Naderi, B.: Modelling the problem of production scheduling for reconfigurable manufacturing systems. 2014. 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '14. Vol. 33, pp.76-80 (2014)
7. Aziz, R. A.: Bound founded answer set programming. In Doctoral Consortium, International Conference on Logic Programming, Vienna (2014).
8. Balduccini, M. : Representing constraint satisfaction problems in answer set programming. In Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), 2009.
9. Baral Ch., Son T.C., Tuan L Ch.: Reasoning about actions in presence of resources: applications to planning and scheduling. In Proceedings of International Conference on Information Technology, pp 3-8, Tata McGraw-Hill (2001).
10. Baral Ch.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press. (2003).
11. Baral, Ch., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. Theory and Practice of Logic Programming (TPLP), Vol.9, Issue 1, pp. 57–144 (2009).
12. Baez-Sentíes, O., Azzaro-Pantel, C., Pibouleau, L. and Domenech, S.: Multi-objective scheduling for semiconductor manufacturing plants, Computers and Chemical Engineering, Vol. 34, Issue 4, pp. 555-566 (2010).
13. Balduccini, M. and Gelfond, M.: Logic programs with consistency-restoring rules, in: Doherty, P., McCarthy, J. and Williams, M.A. (Eds.), International Symposium on Logical Formalization of Commonsense Reasoning, pp. 9 – 18 (2003).
14. Barbosa, J.: Self-organized and evolvable holonic architecture for manufacturing control. Université de Valenciennes et du Hainaut Cambrésis. Retrieved from https://tel.archivesouvertes.fr/tel-01137643. (2015)
15. Bartholomew, M. and Lee, J.: System ASPMT2SMT: Computing ASPMT theories by SMT solvers. In Proceedings of JELIA 2014, pp. 529-542 (2014).
16. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D. A.: Survey of Context Modelling and Reasoning Techniques. Pervasive and Mobile Computing 6, 161-180 (2009).
17. Bhushan, S., & Karimi, I. A.: An MILP approach to automated wet-etch scheduling. Industrial and Engineering Chemistry Research, Vol. 42, Issue 7, pp. 1391–1399 (2003).
18. Biere, A., Heule, M., Van Mareen, H., y Walsh, T.: Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications. IOS Press (2009).
19. Blum, C., J. Puchinger, G. Raidl, A. Roli.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing Vol. 11, Issue 6, pp. 4135–4151 (2011).
20. Bonatti, P., Calimeri, F., Leone, N. and Ricca F.: Answer set programming, in Dovier, A. and Pontelli, E. (Eds.), A 25-Year Perspective on Logic Programming,. Springer-Verlag Berlin, Heidelberg, pp.159-182(2010).
21. Bonfill, A., Camarasa, A.E., Puigjaner, L.: Proactive approach to address the uncertainty in short-term scheduling, Computers & Chemical Engineering, Vol. 32, Issue 8, pp. 1689-1706 (2008).
22. Borgo, S., Leita˜o P.: The role of foundational ontologies in manufacturing domain applications, in: Move to Meaningful Internet Syst. CoopIS, DOA ODBASE, 2004, 670–688 (2004).

23. Brewka, G., Niemelä, I. and Truszczynski, M.: Answer set optimization. In: IJCAI 2003, Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, México, pp.867-872. (2003).
24. Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski.: Preferences and nonmonotonic reasoning. AI Magazine, Vol. 29, Issue 4, pp. 69–78 (2008).
25. Brézillon and S. Abu-Hakima (eds.), Research Report 95/11, LAFORIA, University Paris 6, France, pp. 199-209.
26. Brcic, M., Kalpic, D., Fertalj, K.: Resource constrained project scheduling under uncertainty: a survey. In: CECIIS 2012, Proceedings of the 23rd Central European Conference on Information and Intelligent Systems, Varazdin, Croatia, pp.401- 409,(2012).
27. Brucker P., Drexl A., Möhring R., Neumann K., Erwing P. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. European Journal of Operational Research Vol.112, pp. 3 – 41 (1999) .
28. Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. Theory and Practice of Logic Programming (TPLP), pp. 783–817, November 2015.
29. Bonfill, A., Camarasa, A.E., Puigjaner, L.: Proactive approach to address the uncertainty in short-term scheduling, Computers & Chemical Engineering, Vol. 32, Issue 8, pp. 1689-1706 (2008).
30. Chen, T.:A self-adaptive agent-based fuzzy-neural scheduling system for a wafer fabrication factory. Expert Systems with Applications, Vol. 38, pp. 7158–7168 (2011).
31. Chien, C., Dauzère-Pérès, S., Ehm, H., Fowler, J.W., Jiang, Z., Krishnaswamy, S., Mönch, L. and Uzsoy, R.: Modeling and analysis of semiconductor manufacturing in a shrinking world: challenges and successes. In: WSC 2008, Proceedings of the Winter Simulation Conference, Miami, FL, USA, pp. 2093-2099 (2008).
32. Chien, C., Wu, C. and Chiang Y.: Coordinated capacity migration and expansion planning for semiconductor manufacturing under demand uncertainties, International Journal of Production Economics, Vol. 135. Issue 2, 860-869 (2012).
33. Church, L. and Uzsoy, R.: Analysis of periodic and event-driven rescheduling policies in dynamic shops, International Journal of Computer Integrated Manufacturing, Vol. 5, Issue 3, pp. 153-163 (1992).
34. Cook, S. “The Complexity of Theorem Proving Procedures,” Proc. Third Annual ACM STOC Symp. (1971), 151-158.
35. Coron, J.M., Kawski, M., Zhiqiang, W.: Analyisis of a conservation law modeling a highly re- entrant manufacturing system, Journal of Industrial and Management Optimization, 14(4), 1337-1359 (2010).
36. Crama, Y. J., v.d. Klundert. 1997. “Robotic Flowshop Scheduling is Strongly NP-Complete”. Research Memoranda 010, Maastricht, Maastricht Research School of Economics of Technology and Organization: http://edata.ub.unimaas.nl/wwwedocs/loader/file.asp?id=457.
37. Church, A., "A note on the Entscheidungsproblem", Journal of Symbolic Logic, 1 (1936), pp 40–41.
38. Dantzig G, Orden A, Wolfe P (1955) A generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific J. Math. 5(2):183–195.
39. G.B. Dantzig, On the significance of solving linear programming problems with some integer variables, Econometrica 28 (1960) 30–44.

40. Davenport, A., Beck, Ch. Survey of Techniques for Scheduling with Uncertainty. 2000. unpublished paper. Downloaded in May-17-2016 frome the site http://tidel.mie.utoronto.ca/publications.php.
41. Davenport, A.: Integrated maintenance scheduling for semiconductor manufacturing. In: CPAIOR 2010, Proceedings of the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 6140 of LNCS, (pp.92-96), Bologna, Italy (2010).
42. Davis, Martin; Putnam, Hilary (1960) A Computing Procedure for Quantification Theory. Journal of the ACM. **7** (3): 201–215. doi:10.1145/321033.321034.
43. Davis, M., Logemann, G. and Loveland, D. A machine program for theorem-proving. Communications of the ACM, 5(7):394–397, 1962.
44. Davis, E., & Marcus, G. (2015). Commonsense Reasoning and Commonsense Knowledge in Arti- ficial Intelligence. Communications of the ACM , 58 (9), 92–103.
45. De Cat, B., Denecker, M., and Stuckey, P. 2012. Lazy model expansion by incremental grounding. See Dovier and Costa (2012), 201–211.
46. Dechter, R. and Rish, I. (1994). Directional resolution: the Davis-Putnam procedure, revisited. Proc. KR-94, Bonn, Germany.
47. Delgrande J.P. and Gupta, A. Two results in negation-free logic. Applied Mathematics Letters, 6(6):79–83, 1993.
48. Diedrich, C., Hadlic, T., Riedl, M., Meier, M., Zipper, H., Süss, S. Integration of mechanical information in device descriptions as part of CPS. 2015. International Conference on Computing Techniques and Mechanical Engineering (ICCTME 2015), Phuket, Thailand, UR-CPS (Conference Publishing Services), 2015.
49. Dilkina, B., Gomes, C. and Sabharwal, A.: Tradeoffs in the complexity of backdoor detection for combinatorial problems. In: CP-07, 13th International Conference on Principles and Practice of Constraint Programming, Vol. 4741 of LNCS, (pp. 256-270), Providence, RI, USA (2007).
50. Downing, N., Feydy T. and Stuckey P.: Unsatisfiable cores for constraint programming. CoRR arxiv.org/abs/1305.1690. (2013) Accessed 16 May 2014.
51. Dubois, D.: Possibility theory and statistical reasoning, Computational Statistics and Data Analysis Journal, 51(1), 47-69 (2006).
52. Dümmler, M. 2004. Modeling and optimization of cluster tools in semiconductor manufacturing. Ph. D. thesis, University of Würzburg.
53. Erdem, E.: Applications of answer set programming in phylogenetic systematics. In Balduccini M. and Son T.C. (Eds.): Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, Vol. 6565 of LNCS, pp.415 -431 (2011).
54. Faber, W., Leone, N., Pfeifer, G. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Proceedings of European Conference on Logics in Artificial Intelligence (JELIA), 2004.
55. Feng, W., Zheng, L. and Li, J.: The robustness of scheduling policies in multi-product manufacturing systems with sequence-dependent setup times and finite buffers, Journal of Computers and Industrial Engineering, 63(4), 1145-1153 (2012).
56. Ferraris, P. Answer sets for propositional theories. 2005. In Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), 119– 131.
57. Ferraris, P. and Lifschitz, V. On the stable model semantics of first-order formulas with

aggregates. In Proceedings of the 2010 Workshop on Nonmonotonic Reasoning, 2010.
58. Fitkov-Norris, E.: Literature overview of approaches for enterprise-wide modelling, simulation and optimization. http://eprints.kingston.ac.uk/23969/. (2010) Accessed 10 February 2014.
59. Forrest, E., Hoanca, B. Artificial Intelligence: Marketing's Game Changer. In T. Tsiakis (Ed.), Trends and Innovations in Marketing Information Systems, IGI Global Publishing, Pennsylvania (USA) (2015).
60. Fortnow, L., *"The status of the* **P** versus **NP** *problem"*. *Communications of the ACM* **52** (9): 78–86. 2009. *doi*:*10.1145/1562164.1562186*.
61. Framiñan, J.M. and Ruiz, R.: Architecture of manufacturing scheduling systems: literature review and an integrated proposal, European Journal of Operational Research, 205(2), 237-246 (2010).
62. Gabrel, V., Murat, C. and Thièle A.: Recent advances in robust optimization and robustness: an overview, European Journal of Operational Research, 235(3), 471- 483 (2014).
63. García-Pedrajas, N., Ortiz-Boyer, D., & Hervás-Martínez, C. (2006). An alternative approach for neural network evolution with a genetic algorithm: Cross Garey, M.R., Johnson, D.S., Tarjan, R.E. The planar Hamilton circuit problem is NP-complete, this Journal, 5 (1976), pp. 704-714.
64. Garey, M.R. and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA, 1979.
65. Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., & Zivny, S. (2014). Backdoors into heterogeneous classes of SAT and CSP. In C.E. Brodley, & P. Stone (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada* (pp. 2652–2658). AAAI Press.
66. Gebser, M., Ostrowski, M., Schaub, T. Constraint answer set solving. In Proceedings of International Conference on Logic Programming (ICLP), pages 235–249, 2009.
67. Gebser M.,Kaminski R., Kaufmann B., Ostrowski M., Schaub T., Thiele S.: A Users Guide to gringo, clasp, clingo, and iclingo. 2010. potassco.sourceforge.net/labs.html.
68. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O. and Schaub, T.: 'Stream reasoning with answer set programming: Preliminary Report. In: KR 2012, Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, pp.613-617, Rome, Italy (2012).
69. Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. Clingo = ASP + control: Preliminary report. CoRR, abs/1405.3694, 2014.
70. Gebser, M., Janhunen, T., Jost, H., Kaminski, R., and Schaub, T. ASP Solving for Expanding Universes. In International Conference on Logic Programming and Nonmonotonic Reasoning, Lexington, USA, 2015.
71. Gelfond, M, and Lifschitz, V.: The stable model semantics for logic programming. In: ICLP 1988, Proceedings of the Fifth International Conference on Logic Programming, pp.1070-1080, Seattle, Washington, USA (1988).
72. Gelfond, M. and Lifschitz, V.: Classical negation in logic programs and disjunctive databases, New Generation Computing, 9(3–4), 365-386 (1991).
73. Gelfond, M., Lifschitz, V., Rabinov, A. What are the limitations of the situation calculus? In Robert Boyer, editor, Automated Reasoning: Essays in Honor of Woody Bledsoe, pages 167–179. Kluwer, 1991.
74. Gelfond, M. and Lifschitz, V.: Representing Actions and Change by Logic Programs, Journal of Logic Programming, 17(2,3,4), 301-323 (1993).

75. Gelfond, M. and Yuanlin Zhang, Y.: Vicious circle principle and logic programs with aggregates, Theory and Practice of Logic Programming, 14(4,5), 587-601 (2014).
76. Ghezail, F., Pierreval, H. and Hajri-Gabouj, S.: Analysis of robustness in proactive scheduling: a graphical approach, Computers & Industrial Engineering, 58(2), 193-198 (2010).
77. Geng, N., Jiang, Z. and Chen, F.: Stochastic programming based capacity planning for semiconductor wafer fab wiDth uncertain demand and capacity, European Journal of Operational Research, 198(3), 899-908 (2009).
78. Ghezail, F., Pierreval, H. and Hajri-Gabouj, S.: Analysis of robustness in proactive scheduling: a graphical approach, Computers & Industrial Engineering, 58(2), 193-198 (2010).
79. Giret, A. and Botti, V.: Holons and agents, Journal of Intelligent Manufacturing, 15(5), 645-659 (2004).
80. Goertzel, B., Human-level artificial general intelligence and the possibility of a technological singularity A reaction to Ray Kurzweil's The Singularity Is Near, and McDermott's critique of Kurzweil, Artificial Intelligence 171 (2007) 1161–1173.
81. Gómes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems, Journal of Automated Reasoning, 24(1), 67-100 (2000).
82. Gómes, C. P., Kautz, H., Sabharwal, A. and Selman B.: Satisfiability solvers. In Van Harmelen, F., Lifschitz, V. and Porter B. (Eds.): Handbook of Knowledge Representation (Foundations of Artificial Intelligence), Elsevier, (pp. 89-134). Amsterdam, The Netherlands (2008).
83. Goultiaeva, A. and Bacchus, F. Off the trail: Re-examining the CDCL algorithm. In Proceedings of Theory and Applications of Satisfiability Testing (SAT), pages 30–43, 2012.
84. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: A survey. Annals of Discrete Mathematics 5, 287–326.
85. Giunchiglia, F., Maltese, V., Dutta, B.: Domains and context: First steps towards managing diversity in knowledge. ;J. Web Sem.(2012)53-63.
86. Guha R. V. Contexts: a formalization and some applications. Standford PhD thesis. 1991.
87. Gunnman, S., Eneyo, E. The Structural Integration of Supply Chain Practices in a Manufacturing and Distribution Systems-A Literature Review. International Journal of Emerging Engineering Research and Technology . Volume 4, Issue 1, January 2016, PP 105-118.
88. Gupta JND, Ruiz R, Fowler JW, Mason SJ. Operational planning and control of semiconductor wafer production. Production Planning & Control 2006;17(7): 639–47.
89. Hanks, S. and McDermott, D. Default reasoning, nonmonotonic logics, and the frame problem. Proc. AAAI-86, Philadelphia, PA (1986).
90. Harmelen, F. V., Lifschitz, V. and Porter, B.: Handbook of Knowledge Representation (Foundations of Artificial Intelligence), Elsevier, Amsterdam, The Netherlands (2008).
91. Harrison, J. Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, Cambridge, 2009.
92. Hartmann, S, Briskorn, D. A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research. Vol. 207. Issue 1. November 2010, 1-14.
93. Hartmanis, J., and R. E. Stearns, "On the Computational Complexity of Algorithms." Trans.

Amer. Math. Soc. 117, May 1965.
94. Hartmans, J. Computational Complexity and Mathematical Proofs. Informatics – 10 Years Back, 10 Years Ahead. Reinhard Wilhelm, editor. Springer-Verlag LNCS 2000, 251-256 (2001).
95. Herrmann, J.W. 2006. A history of production scheduling, in: Handbook of Production Scheduling, J.W. Herrmann, ed., Springer, New York.
96. Heljanko, K. and Niemelä, I.: Petri net analysis and nonmonotomic reasoning, Lecture Notes in Computer Science, pp. 7-19 (2000).
97. Herroelen W., Demeulemeester E., De Reyck B., A Classification Scheme for Project Scheduling Problems, Technical Report, Department of Applied Economics, Katholieke Universiteit Leuven, 1997.
98. Hillbert, D. and Ackerman, W. 1928. Grundzüge der theoretischen Logik (Principles of Mathematical Logic). Springer-Verlag, ISBN 0-8218-2024-9.
99. Hitchi High-technologies (VCIM solutions for semiconductor and FPD fabrication lins) http://www.hitachi-hightech.com/global/product_detail/?pn=ict_008.
100. Hoeve, W.: Over-constrained problems. In Milano, M. and Van Hentenryck, P. (Eds.): Hybrid Optimization: the 10 Years of CPAIOR, pp.191-225, Springer (2011).
101. Hölldobler, S., Manthey, N., Philipp, T., Steinke. P. Generic CDCL - A formalization of modern propositional satisfiability solvers. In Daniel Le Berre, editor, 5th Workshop on Pragmatics of SAT — POS 2014, volume 27 of EPiC Series, pages 89–102. EasyChair, 2014.
102. Huang, Y., Zheng, L., Williams, B., Tang, L. and Yang H.: Incremental temporal reasoning in job shop scheduling repair. In: IEEM 2010, Proceedings of the International Conference on Industrial Engineering and Engineering Management, pp.1276-1280. Macau, China (2010).
103. Huang, H.H., Pei, W., Wub, H.H. and May, M.D.: A research on problems of mixed-line production and the re-scheduling, Journal of Robotics and Computer-Integrated Manufacturing, 29(3), 64-72 (2013).
104. Hung, Y.-F., Liang, C.-H., Chen, J. C.: Sensitivity search for the rescheduling of semiconductor photolithography operations, The International Journal of Advanced Manufacturing Technology, 67(1-4), 73-84 (2013).
105. Janhunen, T., Liu, G., Niemelä, I. Tight integration of non-ground answer set programming and satisfiability modulo theories. In Working notes of the 1st Workshop on Grounding and Transformations for Theories with Variables, 2011.
106. Jensen B. Context: A Real Problem for Large and Shareable Knowledge Base. In Proceedings of the International Conference on Building and Sharing of a Very Large-Scale of Knowledge Bases. Tokyo: KB and KS. 1993.
107. Johnson, D.S. Approximation algorithms for combinatorial problems. J. Comput. System Sci., 9 (1974), pp. 256–278.
108. Johnson, D.S. A theoretician's guide to the experimental analysis of algorithms, in: D. S. J. M. H. Goldwasser, C. C. McGeoch (Eds.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, American Mathematical Society, Providence, RI, 2002, pp. 215–250.
109. Johnson, D. A brief history of NP-completeness, 1954–2012. Grötschel M, ed. Documenta Mathematica—Optim. Stories. 21St Internat. Sympos. Math. Programming (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 359–376.
110. Jourdan, L., Basseur, M., and Talbi, E.G. "Hybridizing exact methods and metaheuristics:

A taxonomy," Eur. J. Oper. Res., vol. 199, no. 3, pp. 620–629, 2009.
111. Kanazawa, S.: General Intelligence as a Domain-Specific Adaptation. Psychological Review . 2004, Vol. 111, No. 2, 512–523.
112. Karp, R.M. Combinatorics, Complexity and Randomness. Com of ACM 29, 98-109, 1986.
113. Katragjini, K., Vallada, E., Ruiz, R.: Flowshop rescheduling under different types of disruption, The International Journal of Production Research, 50(1), 780-797 (2013).
114. Kautz, H., Selman, B. (2007)The state of SAT. Discrete Appl. Math. 155:1514–1524.
115. Kieroński, E. and Otto, M. Small substructures and decidability issues for first-order logic with two variables. In Proc. of LICS2005, pages 448–457, 2005.
116. Knuth, D.E.: backus normal form vs. backus naur form. Commun. ACM 7(12), 735–736 (1964).
117. Kronegger, M., Ordyniak, S., Pfandler, A. Variable-deletion backdoors to planning. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25 - 30, 2015, Austin Texas, Austin, USA., pages 2300–2307. AAAI Press, 2014.
118. Kumar, P. R. (1994) Scheduling semiconductor manufacturing plants. IEEE Control Syst pp 33–40.
119. Kumar, V., Kumar, S., Tiwari, M.K., Chan, F.T.S.: Performance evaluation of FMS under uncertain and dynamic situations, Journal of Engineering Manufacture, 222(7), 915-934 (2008).
120. Kuster, J., Jannach, D. and Friedrich, G.: Applying local rescheduling in response to schedule disruptions, Annals of Operations Research, 180(1), 265-282 (2010).
121. Larsen, R. and Pranzo, M.: A framework for dynamic rescheduling problems. Technical report, Dip. Ingegneria dellInformazione, University of Siena, Siena. www.dii.unisi.it/priv/papers/papers_doc/55.pdf. (2012) Accessed 13 September 2012.
122. Lee, Y.F., Jiang, Z.B. and Liu, H.R.: Multiple-objective scheduling and real-time dispatching for the semiconductor manufacturing system, Computers & Operations Research, 36(3), 866-884 (2009).
123. Lee, J. and Meng, Y. On reductive semantics of aggregates in answer set programming. In Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 182–195, 2009.
124. Leitão, P.: Self-organization in manufacturing systems: challenges and opportunities. In: SASOW 2008, Proceedings of the 2nd. International Conference on Self-Adaptive and Self-Organizing Systems Workshops, IEEE, pp.174-179.Venice, Italy (2008).
125. Leitão, P. and Vrba, P.: Recent developments and future trends of industrial agents. In HoloMAS 2011: Proceedings of the 5th Holonic and Multi-Agent Systems for Manufacturing, Vol. 6867 of LNCS, 15-28. Toulouse, France (2011).
126. Leung, J.Y.T.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 1st ed., Chapman & Hall/CRC, London (2004).
127. Li, Z. and Ierapetritou, M.G.: Process scheduling under uncertainty: review and challenges, Computers & Chemical Engineering, 32(4), 715-727 (2008).
128. Lifschitz, V.: Answer set programming and plan generation. Artif. Intell. 138(1-2), 39–54 (2002).
129. Lifschitz, V. What is answer set programming? In Proceedings of the AAAI Conference on Artificial Intelligence, pages 1594–1597, 2008.
130. Lin, J. and Long, Q.: Development of a multi-agent-based distributed simulation platform for semiconductor manufacturing, Expert Systems with Applications, 38(5), 5231-5239 (2011).

131. Liu, H., Jiang, Z. and Fung, R.: Performance modeling, real-time dispatching and simulation of wafer fabrication systems using timed extended object-oriented petri nets, Computers & Industrial Engineering, 56(1), 121-137 (2009) .
132. Liu, S. and Shih, K. Ch.: Construction rescheduling based on a manufacturing rescheduling framework, Automation in Construction, 18(6), 715-723 (2009).
133. Lombardi, M. and Milano, M.: Optimal Methods for Resource Allocation and Scheduling: a Cross-Disciplinary Survey, Constraints, 17(1), 51-85 (2012).
134. Lou, P., Liu, Q., Zhou, Z. and Wang, H., Sun, S.X.: Multi-agent-based proactive–reactive scheduling for a job shop, International J,ournal of Advanced Manufacturing Technology, 59(1 – 4), 311-324 (2012).
135. Luckham, D. The resolution principal in theorem proving. In N. L. Collins and D. Michie, editors, Machine Intelligence, 1, pages 47– 61. Olliver and Boyd, 1967.
136. Ma, Y., Chu, C. and Zuo, C.: A Survey of scheduling with deterministic machine availability constraints, Computers and Industrial Engineering, 58(2), 199-211 (2010).
137. Maratea, M., Pulina L., Ricca, F. Advances in Multi-engine ASP Solving. 2015. Advances in Artificial Intelligence. Vol. 9336 of the series Lecture Notes in Computer Science. pp. 179-190.
138. Marek, V. W. and Truszczynski, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V. W., Truszczynski, M., and Warren, D. S., editors, The Logic Programming Paradigm. A 25-Year Perspective, pages 375-398. Springer Verlag.
139. Marques-Silva J.P. Sakallah, K. GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers, 48(5):506–521, 1999.
140. Marques-Silva J.P. and Sakallah, K.A. “Boolean satisfiability in electronic design automation,” in Proc. Design Automation Conf., Los Angeles, CA, Jun. 2000, pp. 675–680.
141. McCarthy, J.: Programs with commonsense. In Proceedings of the Symposium on the Mechanization of Thought Processes, (pp.1–15).National Physiology Lab, Teddington, England (1959).
142. McCarthy, John (1987): Generality in Artificial Intelligence, Communications of the ACM. Vol. 30, No. 12, pp. 1030-1035. Also in ACM Turing Award Lectures, The First Twenty Years, ACM Press, 1987.
143. McCarthy J. Notes on Formalizing Context. In Proceedings of the Thirteenth Joint International Conference in Artificial Intelligence. 1993.
144. McDermott, D.: Artificial intelligence and consciousness. In Philip David Zelazo, Morris Moscovitch, and Evan Thompson (eds.) 2007 *The Cambridge Handbook of Consciousness*, Cambridge University Press, pp. 117–150.
145. Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkoski, I. and Fahl, M.: State-of-the-art review of optimization methods for short-term scheduling of batch processes, Computers and Chemical Engineering, 30(6-7), 913-946 (2006).
146. Mingers, J., & White, L. (2010). A review of the recent contribution of systems thinking to operational research and management science. European Journal of Operational Research, 207(3), 1147-1161.
147. Mohan, Ch., & Baskaran, R. (2012). A survey: Ant colony optimization based recent research and implementation on several engineering domain. Expert Systems with Applications, 39, 4618–4627.
148. Mönch, L., Stehli, M., and Zimmermannh, J.: FABMAS: An Agent-based system for production control of semiconductor manufacturing processes. Holonic and Multi-Agent

Systems for Manufacturing. Lecture Notes in Computer Science, Vol. 2744, 258-267 (2003).
149. Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations." Journal of Scheduling, 14(6):583-599.
150. Mönch, L., Lars, L., Fowler, J.W., Dauzère-Pérès, S., Mason, S.J. and Rose, O.: A Survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations, Journal of Scheduling, 14(6)), 583-599 (2011).
151. Mondal,S., Ray, P., Maiti, J. Modelling robustness for manufacturing processes: a critical review. International Journal of Production Research, 0 V. 52, n. 2, p. 521-538, 2014.
152. Muñoz, E., Capón-García, E., Moreno-Benito, M., Espuña, A. and Puigjaner, L.: Scheduling and control decision-making under an integrated information environment, Computers & Chemical Engineering, 35(5), 774-786 (2011). Murata T. Petri Nets: Properties, Analysis and Applications. Proc. Of IEEE 77(4). 1989.
153. Nash, J.C. The (Dantzig) simplex method for linear programming. Computing in Science and Engineering, 2(1):29–31, 2000. ISSN 1521-9615. Cited on p. 6.
154. Negri, E., Fumagalli, L., Garetti, M., Tanca, L. Requirements and languages for the semantic representation of manufacturing systems. Computers in Industry. 2015. Article in Press.
155. A. Newell, J.C. Shaw, H.A. Simon. 1963. "Empirical Explorations of the Logic Theory Machine", in Fiegenbaum and Feldman (eds.), Computers & Thought, pp. 134-152, McGraw-Hill, 1963.
156. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R. and Barry, M.: A prolog decision support system for the space shuttle. In PADL 2001: Proceedings of the 3rd International Symposium on Practical Aspects of Declarative Languages, Vol. 1990 of LNCS, Springer, pp.169-183. Las Vegas, Nevada, USA (2001).
157. Novas, J.M. and Henning, G.P.: Reactive scheduling framework based on domain knowledge and constraint programming, Computers and Chemical Engineering, 34(12), 2129-2148 (2010).
158. Ostrowski, M. Schaub, T. ASP modulo CSP: The clingcon system. Theory and Practice of Logic Programming 12, 4-5, 485–503.
159. Ostrowski, M. What is this thing called "clingcon"? A language description. Available at potassco.sourceforge.net.
160. Ostrowski, M. and Schaub, T. 2012. ASP modulo CSP: The clingcon system. TPLP 12, 4-5, 485–503.
161. Ouelhadj, D. and Petrovic, S.: A Survey of dynamic scheduling in manufacturing systems, Journal of Scheduling, 12(4), 417-431 (2009).] (Papadimitriou, 1977).
162. Papadimitriou, C. "The Euclidean traveling salesman problem is NP-complete," Theor. Comp. Sci., vol. 4, no. 3, pp. 237–244, Jun. 1977.
163. Papadimitriou, C. H.: Games against nature, Journal of Computer and System Sciences, 31(2), 288-301 (1985).
164. Papadimitriou, C. H.: Games against nature, Journal of Computer and System Sciences, 31(2), 288-301 (1985).
165. Papadimitriou, C. NP-completeness: A retrospective. *ICALP 97*. (published by Springer LNCS).
166. Patkos T.: A Formal Theory for Reasoning about Action, Knowledge and Time. PhD Dissertation (2010), Department of Computer Science, University of Crete, Greece.

167. Pei, J., Pardalos, P.M., Liu, X., Fan, W., Yang, Sh., Wang, L.: Coordination of production and transportation in supply chain scheduling, Journal of Industrial and Management Optimization, 11(2), 399-419 (2015).
168. Pelov, N., Denecker, M., Bruynooghe, M. Well-founded and stable semantics of logic programs with aggregates. TPLP, 7(3):301–353, 2007.
169. Pham, D.N., Thornton, J.R. and Sattar, A. Modelling and solving temporal reasoning as propositional satisfiability. In Proceeding of the 4th International Workshop on Modelling and Reformulating, Sitges, Spain, 2005.
170. Plümper, T., Troeger, V.E. & Neumayer, E., 2011. Case Selection and Causal Inference in Qualitative Research. Paper presented at the Social Science Research Centre Berlin. Berlin.
171. Popescu, C, Cavia-Soto, M. and Martinez-Lastra, J. L.: Runtime modeling of flow for dynamic deadlock-free scheduling in service-oriented factory automation systems, IETE Tech Rev, 26(3), 203-212 (2009).
172. Powell W (2009) What you should know about approximate dynamic programming. Naval Res. Logist. 56(3):239–249.
173. Proudfoot, D.: Anthropomorphism and AI: Turing's much misunderstood imitation game. Artificial Intelligence 175 (2011) 950–957.
174. Qiao, F., Ma, Y., Li, L. and Yu, H.: A Petri net and extended genetic algorithm combined scheduling method for wafer fabrication, IEEE Transactions on Automation Science and Engineering, 10(1), pp.197-204 (2013).
175. Ramírez, J. and Antonio de, A.: Checking the consistency of a hybrid knowledge base system, Knowledge Based Systems, 20(3), 225-237 (2007).
176. Reportesnreports) http://www.reportsnreports.com/reports/345394-the-global-semiconductor-market-2015-2020.html.
177. Ricca, F., Grasso, G., Alviano, M., Manna, M. Lio, V. Liritano, S. and Leone, N.: Team-building with answer set programming in the gioia-tauro seaport, Theory and Practice of Logic Programming, 12(3), 361-381 (2012).
178. Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. J. Ass. comput. Mach., 12, 23-41.2), 336–345 (1984).
179. Rolón, M. and Martínez, E.: Agent-based modeling and simulation of an autonomic manufacturing execution system, Computers in Industry, 63(1), 53-78 (2012).
180. Ruiz, R. and Rodríguez, J.A.V.: The hybrid flow shop scheduling problem, European Journal of Operational Research, 205(1), 1-18 (2010).
181. Rushby, J. Automated Formal Methods Enter the Mainstream. Communications of the Computer Society of India, Vol. 31, No. 2, May 2007, pp. 28-32. Formal Methods Special Theme Issue.
182. Sabuncuoglu, I. and Goren, S.: Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research, International Journal of Computer Integrated Manufacturing, 22(2), 138-157 (2009).
183. Samer, M. and Szeider, S.: Constraint satisfaction with bounded treewidth revisited, Journal of Computer and System Sciences, 76(2), pp. 103-114 (2010).
184. Samulowitz, H. 2008. Solving Quantified Boolean Formulas. Ph.D. Dissertation, University of Toronto.
185. Schrader, S., Riggs, W.M. and Smith, R.P.: Choice over uncertainty and ambiguity in technical problem solving, Journal of Engineering and Technology Management, 10(1), 73-99 (1993).
186. Schrijver, A. 2012. On the history of the transportation and maximum flow problems.

Grötschel M, ed. Documenta Mathematica—Optim. Stories, 21st Internat. Sympos. Math. Programming (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 169–180.
187. Siddique, N. and Adeli, H., Nature Inspired Computing: An Overview and Some Future Directions. Cognitive Computation. 2015. Vol. 7. Issue 6. pp. 706-714. Journal of Civil Engineering and Management. Vol. 22. Issue 3, 2016, pp. DOI: 10.3846/13923730.2016.1157095.
188. Shen.Y.D. and Eiter, T. (2016) Evaluating Epistemic Negation in Answer Set Programming. Artificial Intelligence. 237: 115-135.
189. Shipp, Stephanie S., Nayanee Gupta, Bhavya Lal, Justin A. Scott, Christopher L. Weber, Michael S. Finnin, Meredith Blake, Sherrica Newsome, and Samuel Thomas. 2012. Emerging Global Trends in Advanced Manufacturing. Alexandria, VA: Institute for Defense Analyses.
190. Sivakumar, A. I. Optimization of cyclye time and utilization in semiconductor test manufacturing using simulation based, on-line, near-real-time scheduling system. In Proceedings of the Winter Simulation Conference, IEEE Computer Society Press, Los Alamitos, CA, 1999; p. 727.
191. Simons, P., Niemelä, I., Soininen, T. Extending and implementing the stable model semantics. Artificial Intelligence, 138:181–234, 2002.
192. Singer, P. (1993). The thinking behind today's cluster tools. Semiconductor International, 46–51.
193. Sohl, D.L. and Kumar, P.R. Fluctuation smoothing scheduling policies for multiple process flow fabrication plants, Proceedings of Seventeenth IEEE/CPMT International Electronics Manufacturing Technology Symposium, Austin, TX, USA, 1995, pp. 190-198.
194. Soininen, T. and Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: PADL 1999: Proceedings of the 1st International Workshop on Practical Aspects of Declarative Languages, Vol. 1551 of LNCS, (pp.305 – 319). San Antonio, TX, USA (1999).
195. Son, T.C., Pontelli, E. and Le, T.: Two applications of the ASP-prolog system: decomposable programs and multi-context systems. In PADL 2014: Proceedings of the Sixteenth International Symposium on Practical Aspects of Declarative Languages, (pp.87-103), San Diego, CA., USA (2014).
196. Song, Y., Zhang, M.T., Yi, J., Zhang, L., Zheng, L.: Bottleneck station scheduling in semiconductor assembly and test manufacturing using ant colony optimization, IEEE Transactions on Automation Science and Engineering, 4(4), 569-578 (2007).
197. Steunebrink, B. R., and Schmidhuber, J. (2012). "Towards an actual gödel machine implementation: A lesson in self-reflective systems," in *Theoretical Foundations of Artificial General Intelligence*, eds P. Wang and B. Goertzel (New York, NY: Springer-Verlag), 173–196. doi: 10.2991/978-94-91216-62-6_10.
198. Szeider, S. The parameterized complexity of constraint satisfaction and reasoning. In H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, and A. Wolf, editors, INAP 2011, and WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers, volume 7773 of Lecture Notes in Computer Science, pages 27–37. Springer, 2011.
199. Tannert, C., Elvers, H. and Jandrig, B.: The ethics of uncertainty: in the light of possible dangers, research becomes a moral duty, EMBO Report, 8(10), 892-896 (2007).
200. Turing, A. "On computable numbers, with an application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, Series 2, 42 (1936-7), pp 230–265.
201. Valckenaers, P., Van Brussel, H., Bongaerts, L. and Wyns, J.: Results of the holonic control

system benchmark at the KULeuven. In: CIMAT 1994, Proceedings of the Conference Computer Integrated Manufacturing and Automation Technology, Vol. 10-12, (pp.128 - 133).Troy, NY, USA (1994).
202. Van Gelder, A., Ross, A. K. and Schlipf, J. S.: The well-founded semantics for general logic programs, Journal of ACM, 38(3), 620-650 (1991).
203. Verfaillie, G. and Jussien, N.: Constraint solving in uncertain and dynamic environments: a survey, Journal Constraints, 10(3), 253-281 (2005).
204. Vieira, G., Herrman, J. and LAntoniou, G.: Verification and correctness issues for nonmonotonic knowledge bases, International Journal of Intelligent Systems, 12(10), 725-738 (1997).
205. Vieira G., Herrman J., Lin E. Rescheduling Manufacturing Systems: A Framework of Strategies, Policias and Methods. Journal of Scheduling 6: 39-62, 2003.
206. Vonder, S.V.D., Demeulemeester, E., Herroelen, W.: A classification of predictive-reactive project scheduling procedures, Journal of Scheduling, 10(3), 195-207 (2007).
207. Vrba, P., Marík, V. and Kadera, P. MAST: From a toy to real-life manufacturing control. In: SNPD 2012, Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp.428-433. Kyoto, Japan (2012).
208. Wen, H., Fu, L. and Huang, S.: Modeling, scheduling, and prediction in wafer fabrication systems using queueing petri net and genetic algorithm. In: ICRA 2001, IEEE International Conference on Robots and Automation, Vol. 4, (pp.3559 – 3564). Seoul, Korea (2001).
209. Wigderson, A. The Gödel Phenomena in Mathematics: A Modern View. In: Kurt Gödel and the Foundations of Mathematics: Horizons of Truth; Baaz, M.; Papadimitriou, C.H.; Putnam, H.W.; Scott, D.S., Eds.; Cambridge University Press: Cambridge, 2010.
210. Wilkins, D. E., Hierarchical Planning: Definition and Implementation, Proceedings 7th ECAI, July 1986.
211. Yao, S., Jiang, Z., Li, N., Zhang, H. and Geng, N.A.: Multi-objective dynamic scheduling approach using multiple attribute decision making in semiconductor manufacturing, International Journal of Production Economics, 130(1), 125-133 (2011).
212. Zakaria, A. and Petrovic, S.: Genetic algorithms for match-up rescheduling of the flexible manufacturing systems, Computers and Industrial Engineering, 62(2), 670-686 (2012).
213. Zawadzki, E. P., Platzer, A. and Gordon, G. J.: A generalization of SAT and #SAT for robust policy valuation. In: IJCAI 2013, Proceedings of the International Joint Conference in Artificial Intelligence, pp.2583-2589, Beijing, China (2013).
214. Zeballos, L. J., Castro, P. M. and Méndez, C. A.: Integrated constraint programming scheduling approach for automated wet-etch stations in semiconductor manufacturing, Industrial and Engineering Chemistry Research, 50(3), 1705-1715 (2011).
215. Zhang, L., Madigan, C.F., Moskewicz, M.W. and Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD 2001, Proceedings of the International Conference of Computer-Aided Design, (pp.279-285). San José, CA, USA (2001).
216. Zhang, D.Z. and Anosike, A.I. (2012). Modeling and simulation of dynamically integrated manufacturing systems, Journal of Intelligent Manufacturing, 23(6), 2367-2382.in, E.: Rescheduling manufacturing systems: a framework of strategies, policies and methods, Journal of Scheduling, 6(1), 39-62 (2003).
217. Zhao, F., Hong, Y., Yu, D., Yang, Y. & Zhang, Q., (2010). A hybrid particle swarm optimisation algorithm and fuzzy logic for process planning and production scheduling integration in holonic manufacturing systems. International Journal of Computer Integrated