Capítulo 1

Introducción a la Optimización

1.1. Introducción

La optimización es una disciplina fundamental en campos de la ciencia tales como la Informática, la Inteligencia Artificial o la Investigación Operativa. En otras comunidades científicas, la definición de optimización se torna bastante impreciso [71], y se relaciona con la idea de *"hacerlo mejor"*. En el marco de este libro, el concepto de optimización se concibe como *el proceso de intentar encontrar la mejor solución posible a un problema de optimización, generalmente en un tiempo limitado.*

Se puede decir que un *problema de optimización* es simplemente un problema en el que hay varias (en general muchas) posibles soluciones y alguna forma clara de comparación entre ellas, de manera que éste existe si y sólo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas. Desde un punto de vista matemático, un problema de optimización P se puede formular como una 3-tupla $P = (f, SS, F)$, definida como sigue:

$$P = \begin{cases} opt: \quad f(x), & \textit{Función Objetivo} \\ s.a., & \\ \quad x \in F \subset SS & \textit{Restricciones} \end{cases} \tag{1.1}$$

donde f es la función a optimizar (maximizar o minimizar), F es el conjunto de soluciones factibles y SS es el espacio de soluciones.

Este tipo de problemas se pueden dividir de forma natural en dos categorías [32][238]: aquéllos en los que la solución está codificada mediante valores reales y aquéllos cuyas soluciones están codificadas por valores enteros. Dentro de la segunda categoría se encuentran un tipo particular de problemas denominados *problemas de optimización combinatoria.* De acuerdo con [238] *un problema de optimización*

combinatoria consiste en encontrar un objeto entre un conjunto finito (o al menos contable) de posibilidades. Este objeto suele ser un número natural (o conjunto de naturales), una permutación o una estructura de grafo (o subgrafo).

Algunos ejemplos muy conocidos de problemas de optimización combinatoria son el problema del viajante de comercio o del agente viajero (TSP) (*Travelling Salesman Problem*), el problema de la asignación cuadrática (QAP) (*Quadratic Assignment Problems*), los problemas de planificación (*Scheduling Problems*) o los problemas de cortes sobre grafos (*Cut Set Problems*). En el capítulo 2 se puede encontrar una descripción de estos problemas (para descripciones más detalladas, pueden consultarse [238][70]).

Los problemas combinatorios presentan como particularidad que siempre existe un algoritmo exacto que permite obtener la solución óptima. Este método consiste en la exploración de forma exhaustiva del conjunto de soluciones (enumeración) [238][70]. Este algoritmo suele ser extremadamente ineficiente, ya que para la mayoría de los problemas de interés que se pueden encontrar en el ámbito de la optimización combinatoria el tiempo que emplearía en encontrar una solución crece de forma exponencial con el tamaño del problema. En general, este tipo de problemas se caracterizan por tener espacios de soluciones con una cardinalidad muy elevada.

Desde finales de los años 70 [114][190] se han intentado caracterizar matemáticamente los problemas combinatorios. De estos estudios se concluye que existe un subconjunto de problemas cuyos algoritmos de resolución presentan una complejidad computacional polinómica, es decir, el tiempo de ejecución de estos algoritmos crece de forma polinómica con el tamaño del problema. Este tipo de problemas se dice que pertenecen a la clase $\mathcal{P}$ y se considera que son resolubles de manera eficiente. Sin embargo, para la mayoría de problemas que tienen interés práctico o científico, no se conoce un algoritmo con complejidad polinómica que lo resuelva de forma exacta. Este tipo de problemas pertenecen a una clase conocida como $\mathcal{NP}$ (En la sección 1.3 se profundiza en estas definiciones). Dado el interés práctico que tiene la resolución de muchos problemas pertenecientes a la clase $\mathcal{NP}$ y la dificultad que existe en resolver dichos problemas de forma exacta, se plantea la opción de encontrar soluciones de alta calidad en tiempos razonables, aunque estas soluciones no sean óptimas. Para esta tarea se necesita utilizar algoritmos aproximados que permitan resolver los problemas combinatorios.

1.1.1. Heurísticas

Para la mayoría de problemas de interés no se conoce un algoritmo exacto con complejidad polinómica que encuentre la solución óptima a dicho problema. Además, la cardinalidad del espacio de búsqueda suele ser enorme, lo cual hace generalmente inviable el uso de algoritmos exactos, fundamentalmente porque la cantidad de

tiempo que se necesitaría para encontrar una solución es completamente inaceptable. Debido a estos dos motivos, se deben utilizar métodos aproximados o heurísticas que permitan obtener una solución de calidad en un tiempo razonable a estos problemas.

El término *heurística* proviene del vocablo griego *heuriskein* que podría traducirse como [196] *encontrar, descubrir o hallar*. La definición que aparece en el *Diccionario de la Real Academia de la Lengua Española* [255] en su segunda y cuarta acepción es la siguiente:

- Técnica de la indagación y del descubrimiento.
- En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.

Por otro lado, la *Enciclopedia Salvat* [284], considera la siguinte definión:

"Arte de inventar o descubrir hechos valiéndose de hipótesis o principios que, aún no siendo verdaderos, estimulan la investigación".

Desde un punto de vista científico, el término heurística se debe al matemático G. Polya quien lo empleó por primera vez en su libro *How to solve it* [249]. Con este término Polya quería expresar las reglas con las que los humanos gestionan el conocimiento común. Actualmente, existen bastantes definiciones para el término heurística. Una de las más claras e intuitivas es la presentada por Zanakis et al. [331]:

"Procedimientos simples, a menudo basados en el sentido común, que se supone que obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido".

Los métodos heurísticos tienen su principal limitación en su incapacidad para escapar de óptimos locales. Esto se debe, fundamentalmente, a que estos algoritmos no utilizan ningún mecanismo que les permita proseguir la búsqueda del óptimo en el caso de quedar atrapados en un óptimo local. Para solventar este problema, se introducen otros algoritmos de búsqueda más *inteligentes* (denominados metaheurísticas [121]) que evitan, en la medida de lo posible, este problema. Este tipo de algoritmos son procedimientos de alto nivel que guían a métodos heurísticos conocidos, evitando que éstos queden atrapados en óptimos locales.

1.1.2. Metaheurísticas

El término *Metaheurística* o *Meta-heurística* fue acuñado por F. Glover en el año 1986 [121]. Etimológicamente, deriva de la composición de dos palabras con origen griego, que son *"meta"* y *"heurística"*. El segundo término ha sido descrito en la sección anterior, mientras que el prefijo meta (en inglés) se podría traducir como *más*

allá de, en un nivel superior. Con este término, Glover pretendía definir un *procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local.* En la literatura se pueden encontrar bastantes definiciones del término *metaheurística* o *heurística moderna* (como se las denomina en [259][211]), que están basadas en la definición original de Glover. Una de las definiciones más descriptivas es la presentada por J.P. Kelly et al. [174], que se puede enunciar del siguiente modo:

"Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos".

La evolución de las metaheurísticas durante los últimos 25 años ha tenido un comportamiento prácticamente exponencial. Desde las primeras reticencias por su supuesta falta de rigor científico [96] hasta la actualidad, se ha resuelto una gran cantidad de problemas que inicialmente parecían inabordables.

La metaheurísticas incluyen[1] (pero no están restringidas a) métodos tan populares como optimización por colonias de hormigas (ACO), algoritmos evolutivos (EA), donde se incluyen los algoritmos genéticos (GA) y los algoritmos meméticos (MA), procedimientos de búsqueda miope (constructiva, voraz o ávida), aleatorizados y adaptativos (GRASP), búsqueda local iterativa (ILS), re-encadenamiento de trayectorias (PR), recocido simulado (SA), búsqueda dispersa (SS) y búsqueda tabú (TS). En el apéndice C se puede encontrar una tabla de acrónimos que contiene el nombre de las metaheurísticas y algunas de las referencias más relevantes. Además los capítulos 5, 7 y 6 están dedicados a describir los aspectos fundamentales de las metaheurísticas más populares.

1.2. Definiciones

Existen tres conceptos básicos que se pueden encontrar en la resolución algorítmica de, prácticamente, cualquier problema de optimización combinatoria. Independientemente de la técnica que se utilice, se debe especificar:

- **Representación:** se encarga de codificar las soluciones factibles para su manipulación. Determina el tamaño (cardinalidad) del espacio de búsqueda (*SS*) de cada problema.

[1] En orden alfabético por su acrónimo en inglés.

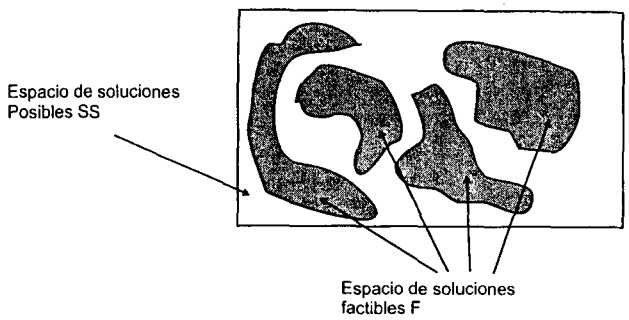

Figura 1.1: Espacio de búsqueda y su subespacio de soluciones factibles.

- **Objetivo:** describe el propósito que se debe alcanzar para una representación dada. Es un predicado matemático que expresa la tarea que se tiene que realizar.
- **Función de evaluación:** permite asociar a cada solución factible un valor que determina su calidad. Es una correspondencia f entre los puntos del espacio de soluciones y $\mathbb{R}$.

Cuando se diseña una función de evaluación[2] se debe tener en cuenta que para muchos problemas no es suficiente con encontrar una solución, sino que ésta debe ser una solución factible; es decir, que satisfaga una serie de restricciones impuestas por el problema. Por lo tanto, se debe establecer una diferencia entre el espacio de búsqueda SS y el espacio de soluciones factibles $F \subseteq SS$. En la figura 1.1 se muestra un ejemplo.

En estas condiciones, se puede definir un *problema de optimización* en los siguientes términos:

Definición 1.1 *Un problema de optimización combinatoria P se puede definir a través de una 3-tupla $P = (f, SS, F)$, donde:*

- *SS se corresponde con un conjunto finito (o infinito contable) de soluciones o configuraciones posibles. Este conjunto se conoce como espacio de búsqueda (search space).*
- *$f : SS \rightarrow \mathbb{R}$ es una función de coste o función objetivo que asigna a cada solución candidata ($x \in SS$) un valor numérico ($f(x) \in \mathbb{R}$).*

[2]Se utilizarán equivalentemente los términos: función de evaluación, función de coste, función objetivo o función de optimización.

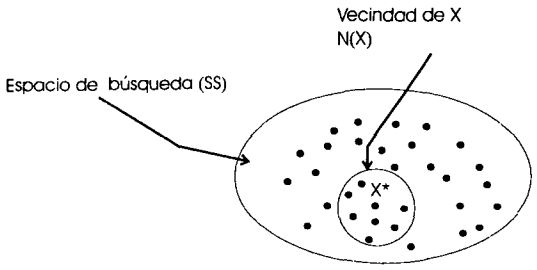

Figura 1.2: Vecindad de una solución x marcado por el área interior a la circunferencia.

- *F determina el conjunto de soluciones factibles $x \in F \subseteq SS$.*

La solución a un problema de optimización (maximización[3]) combinatoria consiste en encontrar un valor de $x^* \in F \subseteq SS$ caracterizado por la siguiente ecuación:

$$x^* \in F \subseteq SS: \quad \forall x \in F \quad f(x^*) \geq f(x) \tag{1.2}$$

1.2.1. Vecindad y óptimos locales

Dada una solución $x \in SS$, la *vecindad* $N(x)$ de esa solución es un subconjunto del espacio de soluciones que contiene soluciones que están *"próximas"* de la solución considerada. En la figura 1.2 se muestra una representación gráfica de la vecindad de una determinada solución x. La cercanía entre dos soluciones del espacio de búsqueda se podría definir de varias formas. A continuación, se describen las más comunes [211]:

- Dado un espacio de búsqueda SS para un problema concreto, se puede definir una función distancia $dist(x,y)$ entre cualquier par de puntos $x,y \in SS$ como sigue:

  $$dist : SS \times SS \rightarrow \mathbb{R}$$

  A partir de esta función se puede definir la vecindad $N(x) \subseteq SS$ de la solución x como:

  $$N(x) = \{y \in SS: \quad dist(x,y) \leq \varepsilon\}$$

[3]La adaptación a problemas de minimización es directa sin más que sustituir " $\geq$" por " $\leq$".

para algún $\varepsilon \geq 0$. Lógicamente, la función distancia descrita en la aplicación anterior depende del problema que se quiera resolver. Por ejemplo, si el espacio de búsqueda fuese euclídeo, la distancia entre dos soluciones $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ se podría calcular como sigue:

$$dist(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

En este caso, la vecindad $N(x)$ de la solución x está formada por todas las soluciones interiores a una hiper-esfera con radio máximo dado por $dist(x, y_{max})$, donde y_{max} representa el punto más lejano que se puede alcanzar.

- Dado un espacio de búsqueda SS para un problema concreto, se puede definir una función N del espacio de soluciones como sigue:

$$N : SS \rightarrow 2^{SS}$$

donde esta función define una vecindad $N(x)$ para cada punto $x \in SS$.

En la vecindad de una solución se encuentran todas aquellas soluciones *"cercanas"* de forma que, dada una solución $x \in N(x)$, cada solución de su vecindad $y \in N(x)$ puede alcanzarse (obtenerse) directamente desde x mediante una operación llamada *movimiento*. Una vez introducidos estos conceptos, se puede definir respectivamente el *óptimo global* y el *óptimo local* de la siguiente manera:

Definición 1.2 *Dado un problema de optimización (f, SS, F), se dice que una solución factible $x \in F \subseteq SS$ es un óptimo (máximo) global si:*

$$\forall y \in F \quad f(x) \geq f(y)$$

Definición 1.3 *Dado un problema de optimización $P = (f, SS, F)$ y una estructura de vecindad N, se dice que una solución factible $x \in F \subseteq SS$ es un óptimo (máximo) local con respecto a N si:*

$$\forall y \in N(x) \quad f(x) \geq f(y)$$

Las definiciones anteriores se pueden particularizar para problemas de minimización sin más que cambiar "$\geq$" por "$\leq$". En la figura 1.3 se representan gráficamente el máximo global, el máximo local y la vecindad, para una función objetivo dada.

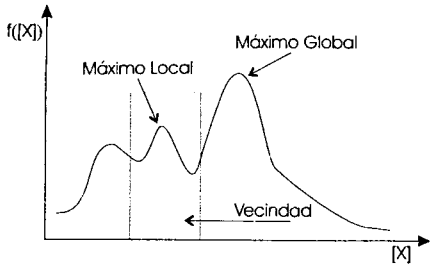

Figura 1.3: Función objetivo con máximo global, máximo local y vecindad.

1.2.2. Intensificación y diversificación

La *intensificación* y la *diversificación* son ideas que se desarrollaron originalmente en el marco de la búsqueda tabú [121][122][123][127]. Actualmente, se han extendido al resto de metaheurísticas como una medida de su *"potencialidad"*:

- La intensificación es la forma de definir el proceso de búsqueda más exhaustivo que puede llevar a cabo una metaheurística en una vecindad dada. Generalmente, la metaheurística no intensifica en cualquier entorno, ya que eso sería una búsqueda exhaustiva de todo el espacio de soluciones, sino que tiene en cuenta factores como, por ejemplo, la calidad de las soluciones encontradas en esa vecindad. Es decir, si en esa vecindad no hay ninguna solución de calidad no parece interesante intensificar la búsqueda en dicha región.

- La diversificación es la forma de definir el proceso mediante el cual la metaheurística es capaz de visitar diversas vecindades lejanas. Habitualmente, la metaheurística no diversifica constantemente y sin criterio, ya que eso sería una búsqueda aleatoria en el espacio de soluciones, sino que tiene en cuenta factores, como por ejemplo el hecho de que una región no haya sido visitada.

Para la mayoría de los problemas, las estrategias de intensificación y la diversificación son contrapuestas. En otras palabras, una metaheurística, cuanto más tiempo le dedique a intensificar la búsqueda en una región dada, menos tiempo le podrá dedicar a buscar en regiones inexploradas. Lógicamente este razonamiento también es cierto a la inversa.

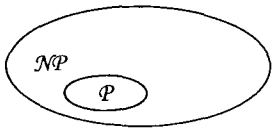

Figura 1.4: Relación entre los problemas $\mathcal{P}$ y $\mathcal{NP}$.

1.3. Complejidad algorítmica: problemas $\mathcal{P}$ y $\mathcal{NP}$

Matemáticamente, los problemas pueden caracterizarse atendiendo a la dificultad que entraña su resolución por un ordenador. Se han definido varias clases de problemas, entre las que destacan las clases $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}-$*completo* y $\mathcal{NP}-$*duro*. En esta sección se introducen definiciones intuitivas y formales para establecer el dominio de cada una de estas clases de problemas.

Se dice que un problema se puede resolver en un tiempo polinómico cuando el tiempo de ejecución de un algoritmo que lo resuelve se puede relacionar con el tamaño de la entrada con una formula polinómica. Los problemas para los que existe un algoritmo polinómico se denominan $\mathcal{P}$. Se considera que los problemas $\mathcal{P}$ se pueden resolver en un tiempo de ejecución razonable para la informática actual.

Una gran cantidad de problemas útiles en el ámbito de la optimización combinatoria son difíciles de resolver. Dicho de otro modo, a día de hoy no se ha encontrado ningún algoritmo que obtenga una solución óptima en tiempo polinómico. Por tanto, no se pueden resolver en un tiempo de ejecución razonable. Este tipo de problemas se pueden categorizar dependiendo de la dificultad de resolución. Hay problemas que, pese a que no se haya encontrado un algoritmo polinómico que los resuelva, si que se puede saber en tiempo polinómico si un valor corresponde a la solución del problema. Por ejemplo, el cálculo de la raíz cuadrada de un número puede ser un problema complicado. En cambio, saber si un determinado valor es la raíz cuadrada de otro es bastante sencillo, ya que basta con elevar ese número al cuadrado. A este tipo de problemas, se les denomina $\mathcal{NP}$. A simple vista, puede parecer que la gran mayoría de los problemas son $\mathcal{NP}$, ya que intuitivamente parece que comprobar una solución es bastante más sencillo que calcularla. Sin embargo, para los problemas de optimización, comprobar si esos valores corresponden a la solución óptima no es nada sencillo. De hecho, existen problemas que requieren ejecutar el mismo algoritmo para buscar soluciones al problema como para comprobar que unos valores son solución.

Los problemas $\mathcal{P}$ también son problemas $\mathcal{NP}$, ya que siempre es posible comprobar que un valor es solución al problema en tiempo polinómico. Si no se encuentra una forma más sencilla de hacerlo, siempre se puede ejecutar el propio algoritmo po-

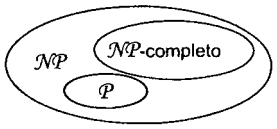

Figura 1.5: Relación entre los problemas $\mathcal{P}$, $\mathcal{NP}$ y $\mathcal{NP}-completo$.

linómico que lo resuelve y comparar el resultado con el valor obtenido. Por tanto, el conjunto de problemas $\mathcal{P}$ es un subconjunto de los problemas $\mathcal{NP}$. En la figura 1.4 se muestra la relación entre los problemas $\mathcal{P}$ y $\mathcal{NP}$.

Por otro lado, existen ciertos problemas de los que siempre se ha pensado que no son $\mathcal{P}$, aunque sí $\mathcal{NP}$, porque nadie ha encontrado nunca un algoritmo polinómico que los resuelva. Sin embargo, en un momento dado, alguien puede encontrar un algoritmo polinómico para un problema considerado $\mathcal{NP}$, de modo que éste pase a ser $\mathcal{P}$. Por ejemplo, hasta 2002 no se encontró un algoritmo polinómico que permite saber si un número es o no primo [3]. Por tanto, se puede decir que para un problema de tipo $\mathcal{NP}$ no se puede obtener una solución en un tiempo razonable hasta que se encuentre un algoritmo polinómico que lo resuelva.

No obstante, hay otro tipo de problemas, llamados $\mathcal{NP}-completos$ que no tienen un algoritmo en tiempo polinómico que los resuelva. No se ha podido demostrar formalmente que no exista, pero los matemáticos creen que realmente no existe. Este tipo de problemas es un subconjunto de los problemas $\mathcal{NP}$, es decir, existe un algoritmo polinómico que puede determinar si un valor es solución al problema. La relación de esta clase de problemas con las demás se muestra en la figura 1.5.

Para saber si un problema es $\mathcal{NP}-completo$, al menos un problema $\mathcal{NP}-completo$ tiene que ser reducible a ese problema. Se dice que un problema A, que es $\mathcal{NP}-completo$, es reducible a otro problema B, cuando se puede crear un algoritmo que resuelva el problema A utilizando como una caja negra un algoritmo para resolver el problema B. Es decir, existe una algoritmo para resolver A de la siguiente forma:

- Toma los datos de entrada del problema A, los transforma de manera que puedan utilizarse como entrada de una caja negra que resuelve el problema B, y la solución a B se pueda transformar a su vez en una solución para el problema A.
- La caja negra que resuelve el problema B puede utilizarse una única vez o un número polinómico de veces dentro del algoritmo que resuelve A.

Considerando que la caja negra que resuelve B se ejecuta en un único paso, el algoritmo para resolver A debe ser polinómico. Todos los problemas $\mathcal{NP}-completos$

Figura 1.6: Relación entre los problemas $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$ – *completo* y $\mathcal{NP}$ – *duro*.

son reducibles entre sí. Es decir, para resolver cualquier problema $\mathcal{NP}$ – *completo* se puede construir un algoritmo polinómico que use una caja negra que, a su vez, resuelve otro problema $\mathcal{NP}$ – *completo*, en caso de que esta caja negra se ejecutase en un único paso. Esto tiene una consecuencia bastante interesante: si se llegara a encontrar un algoritmo polinómico que resuelve cualquier problema $\mathcal{NP}$ – *completo*, todos los demás problemas $\mathcal{NP}$ – *completos* podrían usar ese algoritmo como caja negra. Como el algoritmo que usa la caja negra debe ser también polinómico, existiría un algoritmo polinómico para resolver cualquier problema $\mathcal{NP}$ – *completo*. Dicho de otra forma, como los matemáticos no han encontrado ningún algoritmo polinómico para resolver ninguno de los problemas $\mathcal{NP}$ – *completos*, consideran que no existe tal algoritmo. No obstante, no han podido demostrar matemáticamente que realmente no exista. De hecho, si alguien es capaz de realizar esta demostración o bien encuentra un algoritmo polinómico para algún problema $\mathcal{NP}$ – *completo*, el Instituto Clay de Matemáticas (Cambridge - Massachusetts) le pagaría un millón de dólares.

Finalmente, existe otro tipo de problemas al menos tan difíciles de resolver como los problemas $\mathcal{NP}$ – *completos*, aunque es posible que sean incluso más difíciles. A este tipo de problemas se les denomina $\mathcal{NP}$ – *duros*. Los problemas $\mathcal{NP}$ – *duros* no son un subconjunto de los problemas $\mathcal{NP}$. Es decir, para los problemas $\mathcal{NP}$ – *duros* no existe un algoritmo polinómico que nos permita verificar una solución. Para que un problema sea considerado $\mathcal{NP}$ – *duro*, debe existir un problema $\mathcal{NP}$ – *completo* que sea reducible a ese problema. Es decir, debe existir algún algoritmo polinómico que pueda usar una caja negra que resuelva el problema $\mathcal{NP}$ – *duro* para resolver un problema $\mathcal{NP}$ – *completo*. Se dice que son, al menos, tan difíciles de resolver como un problema $\mathcal{NP}$ – *completo* porque un algoritmo los podría utilizar para resolver un problema NP-completo. Como conclusión, los problemas se pueden categorizar atendiendo a su dificultad en $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$ – *completos* y $\mathcal{NP}$ – *duros*. La relación entre estas clases de problemas se muestra en la figura 1.6.

De forma resumida, Los problemas $\mathcal{P}$ son los problemas para los que existe un algoritmo polinómico que los resuelve. Los problemas $\mathcal{NP}$ son aquellos problemas para los que existe un algoritmo polinómico que verifica una solución. Hay problemas $\mathcal{NP}$ que pueden pasar a ser $\mathcal{P}$ si se encuentra un algoritmo polinómico que los resuelva. Los problemas $\mathcal{NP}$ – *completos* son aquellos problemas de tipo $\mathcal{NP}$ que

los matemáticos consideran que no existe un algoritmo polinómico que los resuelva, aunque no se haya podido demostrar matemáticamente. Y los problemas $\mathcal{NP}$ – *duros* son aquellos al menos tan difíciles como los problemas $\mathcal{NP}$ – *completos*, es decir, que los matemáticos consideran que tampoco existe un algoritmo polinómico que los resuelva.

Rigurosamente hablando, las clases $\mathcal{P}$ y $\mathcal{NP}$ no están compuestas por los problemas de optimización, sino por sus correspondientes problemas de decisión. A continuación se describe la definición de los problemas $\mathcal{P}$ y $\mathcal{NP}$ desde el punto de vista de la Teoría de Lenguajes Formales (En la literatura se pueden encontrar descripciones equivalentes a través de *máquinas de Turing*).

Desde un punto de vista formal, un problema de decisión se define como sigue:

Definición 1 *Dado un lenguaje L de palabras construido sobre un alfabeto* α*, la comprobación de que una palabra x está en L recibe el nombre de problema de decisión.*

Una vez introducido el concepto de problema de decisión se puede definir la clase $\mathcal{P}$ del siguiente modo:

Definición 2 *La clase* $\mathcal{P}$ *esta compuesta por los problemas de decisión (Lenguajes L) para los que existe una función* $f(x)$ *tal que:*
f es polinomial con respecto a la palabra x
$x \in L \Leftrightarrow f(x) = true$
$x \notin L \Leftrightarrow f(x) = false$

Es decir, en esta clase estarían todos aquellos problemas de decisión para los que se puede **encontrar** una solución en tiempo polinómico. Análogamente, se puede definir la clase $\mathcal{NP}$ así:

Definición 3 *La clase* $\mathcal{NP}$ *esta compuesta por los problemas de decisión (Lenguajes L) para los que dada una palabra* $c \in L$ *existe una función* $f(x,c)$ *tal que:*
f es polinomial con respecto a la palabra x
$x \in L \Leftrightarrow f(x,c) = true$
$x \notin L \Leftrightarrow f(x,c) = false$

donde c recibe el nombre de *certificado* y es necesario para comprobar que la cadena x pertenece al lenguaje L.

En esta clase estarían todos aquellos problemas de decisión para los que se puede **comprobar** una solución en tiempo polinómico. Antes de pasar a definir los problemas $\mathcal{NP}$ – *duros*, es necesario introducir el concepto de *lenguaje reducible*:

Definición 4 *Un lenguaje L' se dice que es polinomialmente reducible en tiempo a un lenguaje L si existe una función polinomial f tal que:*
$\forall x \in L' \Leftrightarrow f(x) \in L$

Los problemas $\mathcal{NP}$ – *duros* se caracterizan por ser al menos tan *"difíciles"* como cualquier problema que se encuentre en $\mathcal{NP}$. Formalmente se podrían definir como:

Definición 5 *Un problema de decisión L es $\mathcal{NP}$ – duro ($\mathcal{NP}$ – hard) si:*
$\forall L' \in NP$ *es reducible polinomialmente a L.*

Finalmente, los problemas $\mathcal{NP}$ – *completos* se pueden definir como:

Definición 6 *Un problema de decisión L es $\mathcal{NP}$ – completo ($\mathcal{NP}$ – complete) si es $\mathcal{NP}$ – duro y $\mathcal{NP}$.*

Una consecuencia inmediata que se puede extraer es que, si se encontrase un algoritmo $\mathcal{P}$ para un problema $\mathcal{NP}$ – *duro*, se podría demostrar que $\mathcal{NP} = \mathcal{P}$, ya que cualquier problema $\mathcal{NP}$ se puede reducir polinomialmente a ellos.

1.4. Limitaciones de los algoritmos exactos

Los métodos exactos de resolución de problemas se han aplicado con éxito a una cantidad elevada de problemas. Algunos ejemplos de estos métodos son los algoritmos voraces, algoritmos de divide y vencerás, algoritmos de ramificación y poda, *backtraking*, etc. Todos estos procedimientos resuelven problemas que pertenecen a la clase $\mathcal{P}$ de forma óptima y en tiempo razonable. Como se ha comentado anteriormente, existe una clase de problemas, denominada $\mathcal{NP}$, con gran interés práctico para los cuales no se conocen algoritmos exactos con tiempos de convergencia en tiempo polinómico; es decir, aunque existe un algoritmo que encuentra la solución exacta al problema, tardaría tanto tiempo en encontrarla que lo hace completamente inaplicable. Además, un algoritmo exacto es completamente dependiente del problema (o familia de problemas) que resuelve, de forma que cuando se cambia el problema se tiene que diseñar un nuevo algoritmo exacto y demostrar su optimalidad.