UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIA DE LA COMPUTACIÓN

Evaluación de Algoritmos de Optimización de Primer Orden para la Optimización Topológica Robusta de Dispositivos Nanofotónicos

TESIS

Para optar el título profesional de Licenciado en Ciencia de la Computación

AUTOR:

José Leonidas García Gonzales

ASESOR

Jorge Luis Gonzalez Reaño

Lima - Perú

26 de junio de 2022

Índice general

Pág.

Índice de tablas

Índice de figuras

Capítulo 1

Marco Teórico

En el presente capítulo se introducen los conceptos necesarios para entender la propuesta de esta tesis. Primero, se describe los dos dispositivos a optimizar: (i) *bend* y (ii) WDM. Segundo, se explica como parametrizar estos dispositivos mediante un enfoque basado en píxeles. Tercero, se brindan detalles necesarios para poder simular nuestros diseños utilizando el paquete de Python SPINS-B. Cuarto, se detalla dos transformaciones comúnmente utilizadas en la optimización topológica robusta: (i) filtro por densidad y (ii) proyección. Finalmente, se describe seis algoritmos de optimización que serán utilizados en este trabajo.

1.1 Dispositivos de estudio

1.1.1 *Bend*

Un *bend* es un dispositivo fotónico que se encarga de guiar el giro de un haz de ondas.

En general, al estudiar dispositivos fotónicos es de especial interés la distribución del campo eléctrico ($\boldsymbol{E}$). Este nos permite visualizar la distribución de la energía en un dispositivo calculando lo siguiente:

$$|\boldsymbol{E}|^2 = |\boldsymbol{E_x}|^2 + |\boldsymbol{E_y}|^2 + |\boldsymbol{E_z}|^2, \tag{1.1}$$

donde $\boldsymbol{E_x}, \boldsymbol{E_y}, \boldsymbol{E_z}$ representan las componentes del campo eléctrico en los ejes x, y, z, respectivamente (Lukas Chrostowski, 2010).

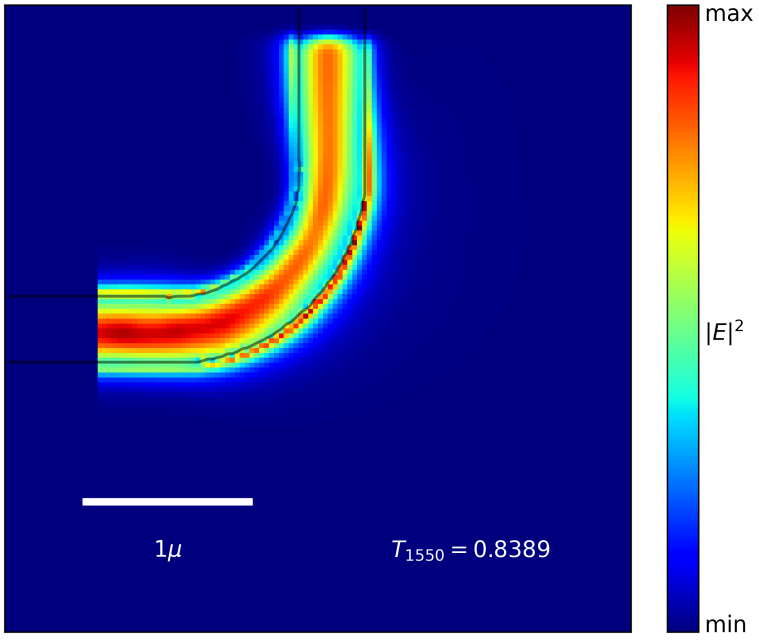


FIGURA 1.1: Representación de $|\boldsymbol{E}|^2$ de un *bend* de $1\mu m$ de radio obtenido usando 3D FDFD bajo una resolución de $30nm$.

Tradicionalmente, un *bend* consiste en una una guía de onda horizontal usada como entrada y una guía de onda vertical usada como salida, estas son conectadas por una guía de onda con la forma de un cuarto de circunferencia de radio r y con el mismo grosor de las guías de onda de entrada y salida. En la Figura 1.1 se muestra un *bend* tradicional de radio $r = 1\mu m$. Como se observa en la imagen, parte significativa de la energía se pierde en la región curva. Esto se debe a que el radio de curvatura es muy pequeño, con un valor más grande (e.g. $r = 10\mu m$) las pérdidas se vuelven casi nulas (Lukas Chrostowski, 2010).

Para evitar ambigüedades, dos puntos adicionales a remarcar son: (i) cuando nos referimos al radio estamos haciendo mención al radio medio de curvatura y (ii) todos los diseños mostrados en este trabajo tienen un profundidad de 220nm.

Observar en una gráfica el valor de $|\boldsymbol{E}|^2$ nos ayuda a entender el funcionamiento de un dispositivo. Por otro lado, una manera de cuantificar que tan bien funciona un diseño es mediante el cálculo de la transmitancia (T). Este valor se define como la relación entre la potencia del flujo que sale del dispositivo con la potencia del flujo que ingresa (Christiansen y Sigmund, 2021b).

Seguidamente, sea λ la longitud de onda de la entrada y $\boldsymbol{P}$ los parámetros que caracterizan al diseño, denotaremos como $T_\lambda(\boldsymbol{P})$ a la transmitancia asociada al dispositivo obtenido con la parametrización $\boldsymbol{P}$ en la longitud de onda λ. Luego, definimos la función objetivo (f_{obj}) para un *bend*, también conocido en el área como figura de mérito (FOM), mediante la siguiente ecuación (Su *et al.*, 2020):

$$f_{obj}(\boldsymbol{P}) = max\left\{T_{1550}(\boldsymbol{P})\right\}. \tag{1.2}$$

En síntesis, la idea detrás de estas definiciones es describir un *bend* mediante una parametrización $\boldsymbol{P}$ (Sección 1.2). Luego, usando algoritmos de optimización, buscar entre las distintas combinaciones de los parámetros aquella configuración que optimice la función f_{obj} (Sección 1.5). De este modo, estaremos encontrando un diseño con una elevada transmitancia, es decir con un buen desempeño.

1.1.2 *Wavelength Demultiplexer* (WDM)

Un WDM es un dispositivo fotónico que se encarga de guiar un haz de ondas de acuerdo a su longitud de onda. Por ejemplo, estos pueden trabajar con dos longitudes de onda y guían las de una longitud por la guía de onda superior y las de otra longitud por la guía de onda inferior.

Análogo al caso del *bend*, utilizaremos la transmitancia para cuantificar el desempeño del dispositivo. Pero, en el presente trabajo usaremos un WDM con dos guías de

salida, por ello, utilizaremos como notación $T_{\lambda}^{(1)}(\boldsymbol{P})$ para representar la transmitancia en la guía de salida superior cuando se recibe un haz de longitud de onda λ en un diseño descrito por los parámetros $\boldsymbol{P}$ y $T_{\lambda}^{(2)}(\boldsymbol{P})$ para la guía de salida inferior.

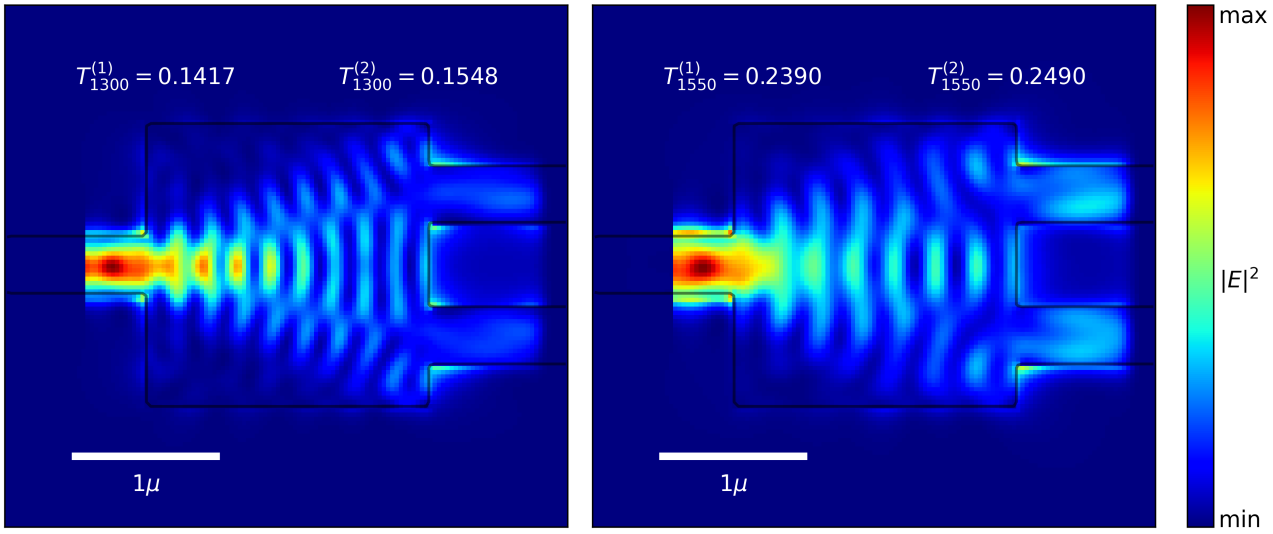


FIGURA 1.2: Representación de $|\boldsymbol{E}|^2$ de un WDM obtenido usando 3D FDFD bajo una resolución de $30nm$.

En la Figura 1.2 se muestra un WDM donde las guías de onda son unidas por una región rectangular de $2.0\mu m \times 2.0\mu m$. En el lado izquierdo se muestra la representación de $|\boldsymbol{E}|^2$ cuando se usa como entrada un haz de ondas de $1300nm$ de longitud, en el lado derecho la entrada es de $1550nm$. En ambos casos el diseño está funcionando más como un *splitter* que como un WDM. Un *splitter* es un dispositivo fotónico que divide la potencia del flujo de la entrada por las guías de salida en una determinada proporción. El diseño intuitivo para este dispositivo es el presentado en la imagen. Similar al caso del *bend*, con unas dimensiones un poco más grandes se puede conseguir pérdidas casi nulas de energía (Lukas Chrostowski, 2010). Sin embargo, como se observa en los valores de la transmitancia, este diseño intuitivo no es apropiado para un WDM.

Basándonos en Su *et al.* (2020), definimos su FOM como

$$f_{obj}(\boldsymbol{P}) = max \left\{ \frac{\left(T^{(1)}_{1300}(\boldsymbol{P})\right)^2 + \left(1 - T^{(2)}_{1300}(\boldsymbol{P})\right)^2 + \left(1 - T^{(1)}_{1550}(\boldsymbol{P})\right)^2 + \left(T^{(2)}_{1550}(\boldsymbol{P})\right)^2}{4} \right\}. \tag{1.3}$$

La Ecuación 1.3 busca maximizar la transmitancia por la guía de onda superior y minimizarla para la guía de onda inferior cuando se recibe una longitud de onda de $1300nm$ y lo contrario para una longitud de onda de $1550nm$. Cabe destacar que la división por cuatro se realiza para asegurar que f_{obj} solamente tenga valores en el intervalo $[0, 1]$, al igual que sucede con la función objetivo del *bend*.

La idea para optimizar un WDM es la misma descrita en la anterior sección. En el resto del capítulo se describe en más detalle los siguientes pasos necesarios para lograr esto.

1.2 Parametrización

Tanto para el *bend* como para el WDM se define una región de diseño mediante una parametrización ($\boldsymbol{P}$) que pueda mapear un gran conjunto de dispositivos. Una de la estrategias más populares para esta tarea es usar parametrización basada en píxeles. Esta estrategia consiste en definir $\boldsymbol{P}$ como una matriz de n filas y m columnas con valores en el intervalo $[0, 1]$. Al proceso de optimizar un dispositivo con esta parametrización se le conoce como optimización topológica (Molesky *et al.*, 2018).

A modo de ejemplo, en la Figura 1.3 se ha definido la matriz $\boldsymbol{P} : [1, n] \times [1, m] \rightarrow [0, 1]$ con $n = m = 18$ y se ha graficado sus valores usando escala gris (0 corresponde al color blanco, 1 al negro y los demás valores a diferentes intensidades del gris). De esta manera observamos como $\boldsymbol{P}$ logra definir la geometría de un diseño. Adicionalmente, conforme se incrementa el valor de n y m se logra definir detalles con mayor precisión.

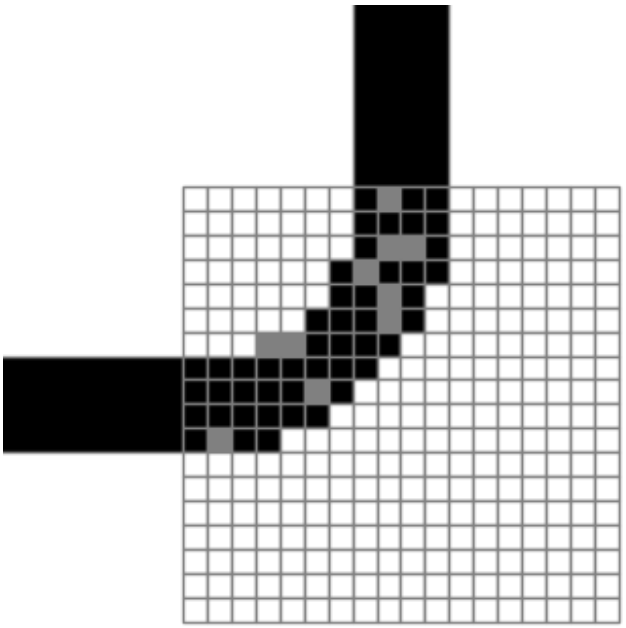


FIGURA 1.3: Parametrización basada en píxeles para un *bend* definiendo $\boldsymbol{P}$ como una matriz de 18×18.

Sin embargo, lo anterior solo nos permite describir la geometría de los diseños, no sus propiedades. Además, no queda claro qué representan las regiones grises. Por ello, para describir las propiedades del diseño es necesario asociar a $\boldsymbol{P}$ alguna propiedad física. Esto lo realizamos calculando la permitividad ($\boldsymbol{\varepsilon}$) mediante la siguiente ecuación:

$$\boldsymbol{\varepsilon}(x, y) = \varepsilon_{Si} + (1 - \boldsymbol{P}(x, y))\varepsilon_{SiO_2} \mid x \in [1, n] \wedge y \in [1, m], \tag{1.4}$$

donde $\varepsilon_{Si} = 3.48^2$ es la permitividad del silicio (Si) y $\varepsilon_{SiO_2} = 1.44^2$ es la permitividad del óxido de silicio (SiO_2). De esta manera, la celda ubicada en la fila x, columna y de la geometría descrita por $\boldsymbol{P}(x, y)$ tiene una permitividad de valor $\boldsymbol{\varepsilon}$(x, y).

Con la Ecuación 1.4 estamos asociando a cada rectángulo de la geometría descrita por $\boldsymbol{P}$ un valor de permitividad en el rango $[\varepsilon_{SiO2} = 1.44^2, 3.48^2 = \varepsilon_{Si}]$. De esta manera, $\boldsymbol{P}(x, y) = 1$ describe que el rectángulo ubicado en la posición (x, y) es de Si, un valor

de $\boldsymbol{P}(x, y) = 0$ describe la presencia de SiO_2 y $0 < \boldsymbol{P}(x, y) < 1$ hace referencia a algún material cuya permitividad es $\boldsymbol{\varepsilon}(x, y)$.

No obstante, un inconveniente de lo descrito es que $\boldsymbol{P}$ puede mapear a materiales inexistestes (regiones grises), mayor detalle de esta dificultad se estudia en la Sección 1.4. Por otro lado, con la descripción de la permitividad ya podemos calcular los campos eléctricos con lo cual se puede obtener el valor de f_{obj} definido para el *bend* y WDM. Por esta razón se realizan simulaciones electromagnéticas y se resuelven las inversas de las ecuaciones de Maxwell, esto permite obtener los campos eléctricos (Su *et al.*, 2020). Afortunadamente, para este proceso se puede utilizar distintos programas que implementan diversos métodos numéricos para realizar los cálculos (Sección 1.3).

1.3 Simulación

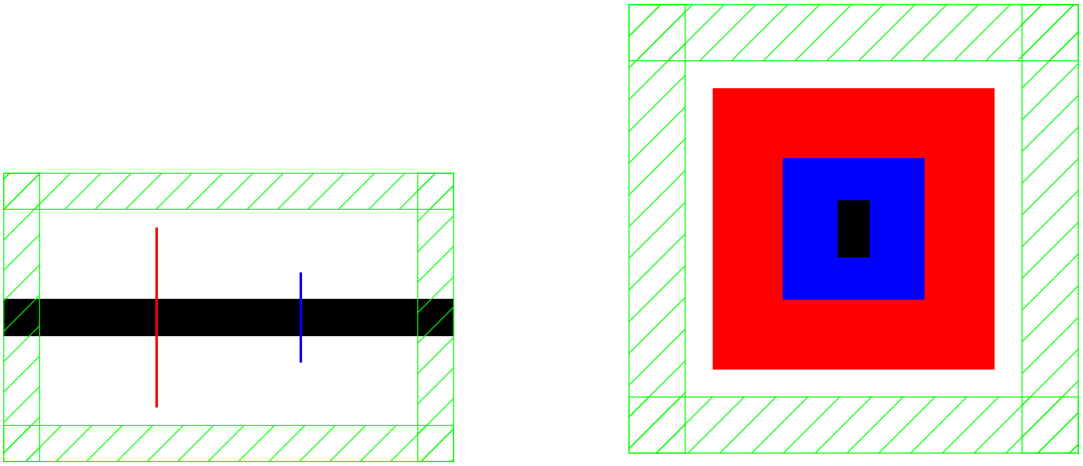
Una vez tenemos definido un dispositivo con regiones fijas (guías de onda) y una región de diseño (descrita por la parametrización), es necesario incorporar tres elementos adicionales (Oskooi *et al.*, 2010, Su *et al.*, 2020):

1. **Fuente:** Suele representarse como un rectángulo en un plano perpendicular al flujo que pasa por el lugar donde este se ubica. Simula la emición de un haz de ondas por el diseño.

2. **Monitores:** Suelen representarse como un rectángulo similar a la fuente. Capturan información en su ubicación (e.g. valores del campo eléctrico).

3. ***Perfectly Matched Layer* (PML):** Representan las condiciones de frontera en la simulación. Se utilizan para limitar el espacio donde se deberá realizar las simulaciones computacionales.

En la Figura 1.4 se ilustra una guía de onda con los elementos mencionados. El rectángulo de color negro representa la guía de onda, la región roja la fuente, la región



(a) Vista frontal (b) Vista lateral

FIGURA 1.4: Configuración de simulación para una guía de onda.

azul el monitor y lo verde el PML. Guiándonos de la figura de la izquierda, al realizar una simulación con esta configuración el flujo iría de la región roja a la región azul. Sin embargo, este seguiría expandiéndose por la guía de onda hasta llegar a la región verde (PML). En palabras sencillas, podemos interpretar el PML como una caja que permite aislar nuestro sistema. Por otro lado, la figura de la derecha representa el mismo diseño pero desde una vista lateral (recordemos que son diseños en 3D). Aquí podemos evidenciar como la fuente y el monitor son representados por rectángulos, no por rectas.

Siguiendo lo descrito, podemos utilizar diversos programas para configurar y ejecutar simulaciones de un *bend* y WDM. Una alternativa es usar SPINS-B junto a Maxwell-B, paquetes de Python de código abierto. Estos paquetes funcionan bajo una arquitectura cliente-servidor donde SPINS-B es el cliente y Maxwell-B es el servidor.

Por un lado, SPINS-B permite configurar diseños (fuente, monitores, PML, parametrización, región de diseño, etc) y solicitar al servidor que obtenga sus propiedades. Un aspecto interesante de este paquete es que trabaja con bloques llamados nodos. Estos representan operaciones fundamentales como suma, resta, multiplicación, etc. Luego,

el programa guarda nuestra función objetivo (f_{obj}) usando un grafo que tiene estos nodos de vértices. Así, SPINS-B permite calcular de forma eficiente (usando un número de simulaciones independiente de las dimensiones de $\boldsymbol{P}$) el valor de $\nabla f_{obj}(\boldsymbol{P})$ mediante diferenciación automática (Kochenderfer y Wheeler, 2019, Su *et al.*, 2020).

Por otro lado, Maxwell-B implementa un método numérico conocido como *finite-difference frequency domain* (FDFD) para resolver numéricamente las ecuaciones de Maxwell y obtener así determinadas propiedades de los diseños que recibe. Es importante destacar que la implementación de esta librería es en GPUs de NVIDIA y permite aprovechar múltiples GPUs a la vez.

Un último detalle a señalar es que para las simulaciones debemos definir su resolución, esto lo representamos con el parámetro dx. Su valor se utiliza para discretizar nuestro espacio de simulación en una malla de cubos de dimensiones $dx \times dx \times dx$. En general, cuanto más pequeño es este valor los resultados son más precisos, pero la cantidad de memoria y el tiempo de simulación se incrementan considerablemente.

1.4 Transformaciones

En esta sección se discute dos transformaciones populares en optimización topológica para imponer restricciones de fabricación en nuestreos diseños: (i) filtro por densidad y (ii) proyección.

1.4.1 Filtro por Densidad

Antes de definir el filtro por densidad, necesitamos definir el concepto de bola cerrada. Para ello, tenemos que definir el concepto de función distancia en nuestra matriz de parametrización. Así, sea $x, x' \in [1, n] \wedge y, y' \in [1, m]$, definimos la distancia entre las posiciones (x, y) y (x', y') de la matriz $\boldsymbol{P}$ como:

$$dis(x, y, x', y') = |x - x'| * psize_y + |y - y'| * psize_x, \tag{1.5}$$

donde $psize_x$ es el valor de la longitud horizontal que representa cada celda de $\boldsymbol{P}$ y $psize_y$ es el recíproco para la longitud vertical.

De esta manera, definimos la bola cerrada ($\overline{B}_r(x, y)$) de centro (x, y) y radio $0 < r$ mediante el siguiente conjunto:

$$\overline{B}_r(x, y) = \{(x', y') | 1 \leq x' \leq n \wedge 1 \leq m \leq m \wedge dis(x, y, x', y') \leq r\}. \tag{1.6}$$

Seguidamente, usando estos conceptos en la matriz $\boldsymbol{P}$ se define el filtro por densidad ($\widetilde{\boldsymbol{P}}$) mediante la ecuación:

$$\widetilde{\boldsymbol{P}}(x, y) = \frac{\sum\limits_{(x',y') \in \overline{B}_{r_f}(x,y)} w_{x,y}(x', y') A(x', y') \boldsymbol{P}(x', y')}{\sum\limits_{(x',y') \in \overline{B}_{r_f}(x,y)} w_{x,y}(x', y') A(x', y')}, \tag{1.7}$$

$$w_{x,y}(x', y') = max(0, r_f - dis(x, y, x', y')), \tag{1.8}$$

donde $A(x, y)$ es el área de la celda representada por $\boldsymbol{P}(x, y)$. Por otro lado, el concepto de mínimo radio de curvatura (r_f) hace referencia a que un diseño se puede dibujar con una circunferencia de radio r_f. El objetivo de la Ecuación 1.8 es tratar de imponer este concepto al diseño.

Finalmente, de la Ecuación 1.7 podemos obtener el valor de su derivada parcial respecto al diseño original calculando lo siguiente:

$$\frac{\partial \widetilde{\boldsymbol{P}}(x,y)}{\partial \boldsymbol{P}(x^*,y^*)} = \frac{w_{x,y}(x^*,y^*)A(x^*,y^*)}{\sum\limits_{(x',y')\in \overline{B}_{r_f}(x,y)} w_{x,y}(x',y')A(x',y')}. \tag{1.9}$$

1.4.2 Proyección

La proyección descrita en la sección anterior ayuda a obtener diseños que eviten regiones punteagudas; sin embargo, esta transformación suele generar regiones grises. Por este motivo, es común acompañar el filtro por densidad con una proyección ($\widetilde{\widetilde{\boldsymbol{P}}}$) descrita como:

$$\widetilde{\widetilde{\boldsymbol{P}}}(x,y) = \frac{\tanh(\beta \times \eta) + \tanh(\beta \times (\widetilde{\boldsymbol{P}}(x,y) - \eta))}{\tanh(\beta \times \eta) + \tanh(\beta \times (1-\eta))}, \tag{1.10}$$

donde η y β son números escalares. Para entender el impacto de estos parámetros en la Ecuación 1.10, veamos la Figura 1.5. En la figura se está graficando esta ecuación con un valor fijo de $\eta = 0.5$ y distintos valores de β. Como podemos observar, con $\beta = 1$ tenemos prácticamente la función identidad y conforme aumenta el valor de β la función se aproxima más a una función escalonada con quiebre en 0.5. En realidad, en general el quiebre se realiza en el valor que se le de a η.

Además, de la Ecuación 1.10 podemos obtener el valor de su derivada parcial respecto al diseño $\widetilde{\boldsymbol{P}}$ mediante la siguiente ecuación:

$$\frac{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x,y)}{\partial \widetilde{\boldsymbol{P}}(x,y)} = \frac{(1 - \tanh^2(\beta \times (\widetilde{\boldsymbol{P}}(x,y) - \eta)))\beta}{\tanh(\beta \times \eta) + \tanh(\beta \times (1-\eta))}. \tag{1.11}$$

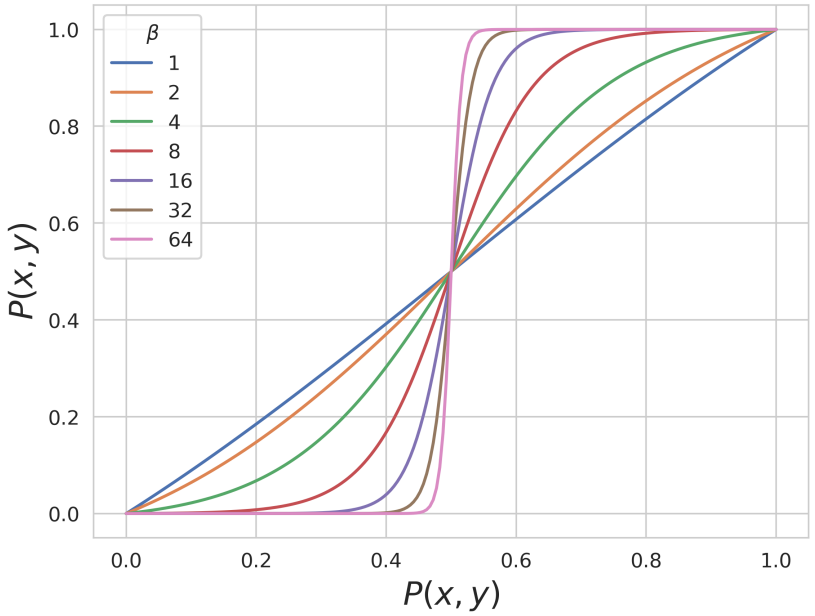
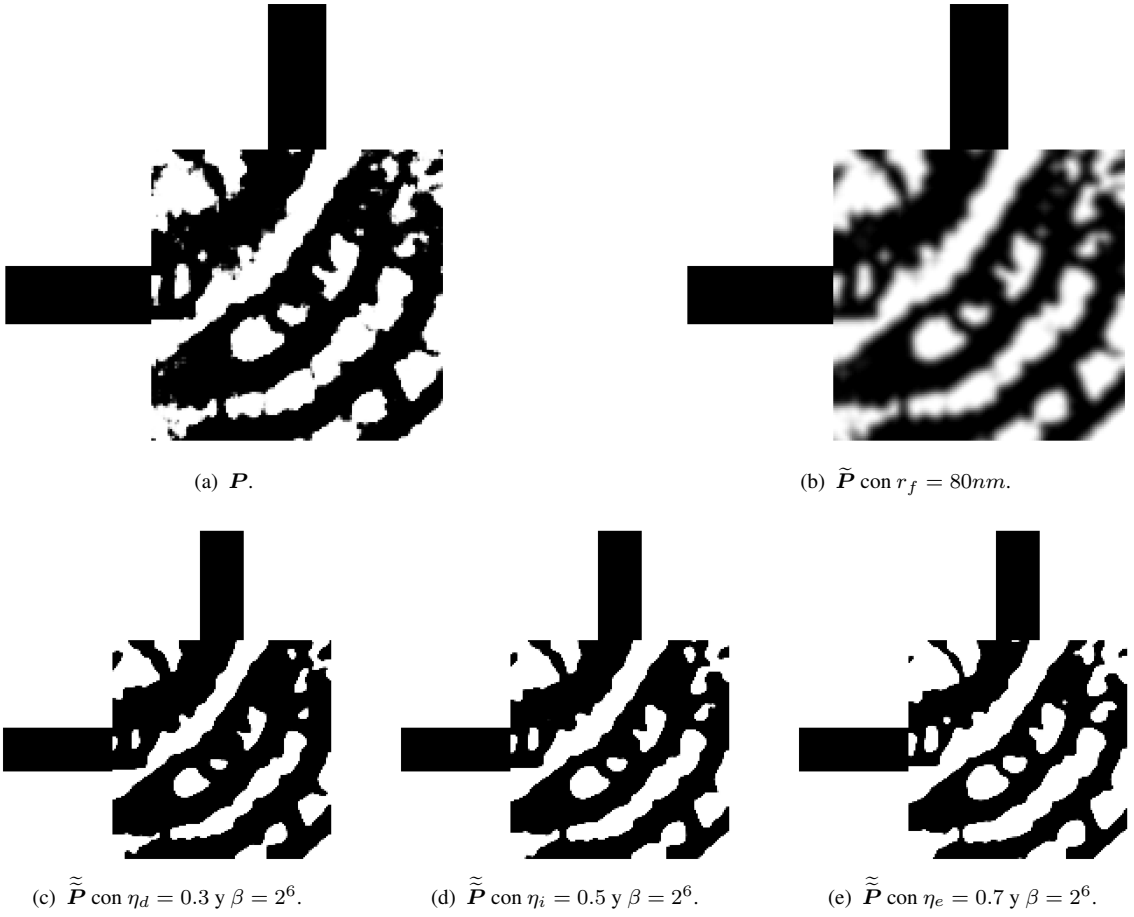


FIGURA 1.5: Transformación de proyección con $\eta = 0.5$ y distintos valores de β.

1.4.3 Aplicación de las Transformaciones

Con el objetivo de visualizar estas transformaciones, observemos la Figura 1.6. En la Figura 1.6(a) se muestra el diseño que representa una parametrización $\boldsymbol{P}$. A su derecha se encuentra la Figura 1.6(b), aquí se aplicó el filtro de densidad utilizando un radio de curvatura $r_f = 80nm$. Como podemos observar, este diseño posee regiones grises, mas ha logrado eliminar zonas punteagudas. Seguidamente, en las Figuras 1.6(c), 1.6(d), 1.6(e) se muestra el diseño de la Figura 1.6(b) tras aplicar la proyección descrita con $\beta = 2^6$ y distintos valores de η. Al utilizar $\eta = \eta_i = 0.5$ (Figura 1.6(d)) estamos simplemente discretizando el diseño. Por otro lado, al usar $\eta = \eta_d = 0.3$ (Figura 1.6(c)) estamos simulando que el diseño se ha dilatado (i.e. la región de Si del diseño intenta expandirse). Finalmente, al usar $\eta = \eta_e = 0.7$ (Figura 1.6(e)) estamos simulando que el diseño ha erosionado (i.e. la región de SiO_2 del diseño intenta expandirse).



(a) $\boldsymbol{P}$. (b) $\widetilde{\boldsymbol{P}}$ con $r_f = 80nm$.

(c) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_d = 0.3$ y $\beta = 2^6$. (d) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_i = 0.5$ y $\beta = 2^6$. (e) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_e = 0.7$ y $\beta = 2^6$.

FIGURA 1.6: Aplicación del filtro de densidad y proyección a una parametrización $\boldsymbol{P}$.

Así, observamos como la proyección presentada en esta sección no solo nos permite discretizar diseños, sino también simular una dilatación o erosión de estos. Particularmente, la optimización topológica se considera robusta cuando tiene en cuenta estas dos posibles modificaciones en el proceso de optimización. Por otro lado, del mismo modo que se aplicó en la Figura 1.6, para poder diferenciar el uso de η denotaremos como η_i al valor usado en un diseño nominal, η_e para los diseños erosionados y η_d para los diseños dilatados. Además, es importante señalar que se suele escoger los valores de η_e y η_d de las manera que $\eta_d = 1 - \eta_e$. Por último, notemos que estos parámetros deben seleccionarse con cuidado; caso contrario, el diseño podría simular una dilatación o erosión muy exageradas.

Un último punto a considerar es que nuestro proceso de optimización necesitará calcular $\nabla f_{obj}(\boldsymbol{P})$, pero al aplicar estas transformaciones terminaremos calculando $\nabla f_{obj}(\widetilde{\widetilde{\boldsymbol{P}}})$. Para solucionar este problema podemos implementar las transformaciones presentadas de manera que aprovechen la diferenciación automática (similar a lo realizado en la implementación de MEEP (Oskooi *et al.*, 2010)); sin embargo, en el presente trabajo optamos por un enfoque más sencillo que consiste en utilizar la Ecuación 1.9 y la Ecuación 1.11 para realizar el cálculo utilizando lo siguiente:

$$\frac{\partial f_{obj}}{\partial \boldsymbol{P}(x,y)} = \sum_{(x',y')\in \overline{B}_{r_f}(x,y)} \frac{\partial f_{obj}}{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x',y')} \frac{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x',y')}{\partial \widetilde{\boldsymbol{P}}(x',y')} \frac{\partial \widetilde{\boldsymbol{P}}(x',y')}{\partial \boldsymbol{P}(x,y)}. \tag{1.12}$$

Para finalizar esta sección, se presenta una función para medir la presencia de regiones grises. Esta función (M_{nd}) nos ayudará en la evaluación de los diseños optimizados obtenidos en esta tesis.

$$M_{nd}(\boldsymbol{P}) = \frac{\sum_{1\leq x\leq n \wedge 1\leq y\leq m} 4\boldsymbol{P}(x,y)(1-\boldsymbol{P}(x,y))}{n\times m} \times 100\,\%. \tag{1.13}$$

Una discusión más detallada de todo lo explicado en esta sección se puede encontrar en Lazarov *et al.* (2016).

1.5 Algoritmos de Optimización

Finalmente, con todo lo descrito en este capítulo, ya tenemos las herramientas necesarias para plantear la optimización de nuestros dispositivos. Ahora, notemos que nuestro problema se termina reduciendo a maximizar f_{obj}. Sin pérdida de generalidad, en la presente sección se describen algoritmos para minimizar la función $f = -f_{obj}$. Esto es posible ya que maximizar una función es equivalente a minimizar su negativo (Kochenderfer y Wheeler, 2019).

Para minimizar la función f, sea $\mathscr{P}$ el conjunto de todas las matrices $\boldsymbol{P} : [1, n] \times [1, m] \rightarrow [0, 1]$, queremos encontrar algún mínimo global $\boldsymbol{P}^* \in \mathscr{P}$, donde este elemento se define como una matriz que satisface que $f(\boldsymbol{P}^*) \leq f(\boldsymbol{P}), \quad \forall \boldsymbol{P} \in \mathscr{P}$. Dicho de otra manera, nuestro problema es encontrar $\boldsymbol{P}^*$; sin embargo, esto es una tarea computacionalmente dificil de resolver (Angeris *et al.*, 2021).

Un problema más viable es encontrar $\sigma > 0 \wedge \boldsymbol{P}^+ \in \mathscr{P}$ tal que $f(\boldsymbol{P}^+) \leq f(\boldsymbol{P}), \quad \forall \boldsymbol{P} \in \mathscr{P} : ||\boldsymbol{P} - \boldsymbol{P}^+|| < \sigma$. Es decir, es más práctico encontrar $\boldsymbol{P}^+$ (conocido como mínimo local) el cual es un elemento que es un mínimo global si restringimos el espacio de búsqueda a cierta región alrededor de este elemento.

Debemos resaltar que ninguno de los algoritmos de esta sección puede asegurar encontrar el mínimo global; sin embargo, sus estrategias de búsqueda suelen permitir encontrar óptimos locales adecuados (Angeris *et al.*, 2021, Schneider *et al.*, 2019). Además, todos estos algoritmos se aseguran de mantener los valores de $\boldsymbol{P}$ en el intérvalo $[0, 1]$ aplicando $\boldsymbol{P}(x, y) \leftarrow max(0, min(1, \boldsymbol{P}(x, y)))$, mas este detalle se obvia en las explicaciones por simplicidad en la notación.

1.5.1 *Genetic Algorithms* (GA)

Como se describe en el Algoritmo 1 (Kochenderfer y Wheeler, 2019), la idea es comenzar generando una población (*population*) de tamaño $population\_size$. Esta población es un conjunto de matrices de $n \times m$ generadas a partir de una parametrización inicial $\boldsymbol{P}$, línea 1. Los siguientes tres pasos se ejecutan por k iteraciones. Primero, se realiza un proceso de selección para obtener las mejores parametrizaciones (*parents*), línea 3. Segundo, los seleccionados se encargan de producir la nueva generación (*children*), línea 4. Tercero, la nueva generación muta obteniendo nuevas características, línea 5.

Como se observa en el Algoritmo algorithm 1, tenemos las siguientes funciones:

Algorithm 1: *Genetic Algorithms* (GA)

Data: $\boldsymbol{P}, population\_size, GA\_range, n\_selected\_parents, prob\_mutation$
Result: $min(population)$

```
population = generate_population()
for t = 0; t < k; t++ do
    parents = select(population)
    children = crossover(parents)
    population = mutation(children)
```

- $generate\_population()$: retorna $population\_size$ matrices $\boldsymbol{P'}$ tal que $|\boldsymbol{P'}(x,y) - \boldsymbol{P}(x,y)| \in U(-GA\_range, GA\_range), \quad \forall x \in [1,n] \wedge y \in [1,m]$.

- $select(population)$: retorna $n\_selected\_parents$ elementos de $population$ de acuerdo a la probabilidad $prob_i$ dada por la ecuación

$$prob_i = \frac{max(f) - f^{(i)}}{\sum_j max(f) - f^{(j)}}, \tag{1.14}$$

  donde $f^{(i)}$ representa el valor de f aplicado a la $i-$ésima parametrización de $population$ y $max(f)$ es el máximo valor de f al haber sido aplicado a todas las matrices de $population$.

- $crossover(parents)$: retorna $population\_size$ nuevas parametrizaciones. Cada nueva matriz $\boldsymbol{S}$ es la combinación de dos parametrizaciones aleatorias $\boldsymbol{P_1}$ y $\boldsymbol{P_2}$ seleccionados de $parents$. En esta combinación, para cada $x \in [1,n] \wedge y \in [1,m]$ se define con igual probabilidad que $\boldsymbol{S}(x,y) = \boldsymbol{P_1}(x,y)$ o $\boldsymbol{S}(x,y) = \boldsymbol{P_2}(x,y)$.

- $mutation(children)$: retorna $children$, pero a cada elemento de todas estas matrices le agrega un valor en $U(-GA\_range, GA\_range)$ con probabilidad $prob\_mutation$.

1.5.2 *Particle Swarm Optimization* (PSO)

Podemos pensar este algoritmo como un caso especial del Algoritmo 1 (Kochenderfer y Wheeler, 2019). La idea es visualizar el $i-$ésimo individuo como una partícula definida por su posición ($\boldsymbol{P}^{(i)}$), velocidad ($\boldsymbol{V}^{(i)}$) y la mejor posición encontrada por la partícula ($\boldsymbol{P}_b^{(i)}$). Para nuestro problema la matriz de parametrización representa la posición y la velocidad es una matriz que guía la exploración a otras parametrizaciones.

La idea principal del algoritmo es que cada particula acumula velocidad en una dirección favorable dada por: (i) la mejor posición encontrada hasta el momento por esta partícula y (ii) la mejor posición encontrada por la población completa. Como consecuencia, los individuos se pueden mover independientemente de perturbaciones locales. Adicionalmente, agregando caminos aleatorios los individuos incorporan comportamientos impredecibles que puede permitirles encontrar potenciales mejores elementos. Esta idea se sintetiza en el Algoritmo 2.

Algorithm 2: *Particle Swarm Optimization* (PSO)

Data: $\boldsymbol{P}, population\_size, PSO\_range, \omega, c_1, c_2$
Result: $\boldsymbol{P_b}$

```
population = generate_population()
for t = 0; t < k; t++ do
    P_b = select(population)
    population = mutation(population, P_b)
```

Como se observa en el algorithm 2, tenemos las siguientes funciones:

- $generate\_population()$: retorna $population\_size$ matrices $\boldsymbol{P'}$ tal que $|\boldsymbol{P'}(x,y) - \boldsymbol{P}(x,y)| \in U(-PSO\_range, PSO\_range), \quad \forall x \in [1,n] \wedge y \in [1,m]$. Además genera las matrices V con valores en $U(0,1)$.
- $select(population)$: retorna el individuo con el menor valor de f encontrado hasta el momento.

- $mutation(population, \boldsymbol{P_b})$: Aplica las siguientes transformaciones a *population*:

$$\boldsymbol{P}^{(i)} \leftarrow \boldsymbol{P}^{(i)} + \boldsymbol{V}^{(i)}, \tag{1.15}$$

$$\boldsymbol{V}^{(i)} \leftarrow \omega \boldsymbol{V}^{(i)} + \ c_1 r_1 \left(\boldsymbol{P}_{\boldsymbol{b}}^{(i)} - \boldsymbol{P}^{(i)} \right) + \ c_2 r_2 \left(\boldsymbol{P_b} - \boldsymbol{P}^{(i)} \right), \tag{1.16}$$

  donde $\boldsymbol{P_b}$ es la mejor posición encontrada globalmente, ω representa la tendencia de la particula de conservar su velocidad actual, c_1 y c_2 cuantifica la atracción relativa con $\boldsymbol{P}_{\boldsymbol{b}}^{(i)}$ y $\boldsymbol{P_b}$ respectivamente, y $r_1, r_2 \in U(0,1)$ representan el comportamiento impredecible.

1.5.3 *Covariance Matrix Adapatation Evolution Strategy* (CMA-ES)

La idea general de esta estrategia evolutiva, mostrada en el Algoritmo 3, es mantener: (i) un vector $\boldsymbol{\mu}$ p-dimensional, (ii) una matriz $\boldsymbol{C}$ y (iii) un número σ para ir generando n inviduos p-dimensionales a partir una distribución $\mathcal{N}(\mu,\ \sigma^2\boldsymbol{C})$.

Tomar puntos de esta distribución limita el espacio de búsqueda a una hiperelipse. Así, el algoritmo evalua puntos en esta región limitada. Luego, usando los valores obtenidos, se puede decidir entre: (i) mover la hiperelipse a otra región del espacio de búsqueda (ii) expandir or reducir la región cubierta por la distribución. El algoritmo de CMA-ES trabaja iterativamente sobre esta idea hasta que la hiperelipse termina casi degenerándose en un punto, potencialmente un óptimo.

Entrando en más detalles del Algoritmo 3, en la línea 1 simplemente se linealiza la matriz $\boldsymbol{P}$ en un vector $\boldsymbol{\mu}$ de dimensión $p = n \times m$ (i.e. se unen sus vectores filas en orden para formar un solo vector). Luego, dentro del bucle en la iteración t, en la línea 3 se genera $population\_size$ puntos p-dimensionales $\boldsymbol{x_i}$ de la distribución $\mathcal{N}(\boldsymbol{\mu},\ \sigma^2\boldsymbol{C})$, donde

Algorithm 3: CMA-ES

Data: $\boldsymbol{P}, population\_size, \sigma$
Result: $\boldsymbol{\mu}$
1 $\boldsymbol{\mu} = flatten(\boldsymbol{P})$
2 **for** $t = 0$; $t < k$; t++ **do**
3 sample() // Obtener *population_size* puntos de $\mathcal{N}(\boldsymbol{\mu}, \sigma^2\boldsymbol{C})$
4 update() // Ecuación 1.17
5 control() // Ecuación 1.18
6 adapt() // Ecuación 1.20

estos son ordenados ascendentemente de acuerdo al valor de f. En la línea 4 actualizamos la media $\boldsymbol{\mu}$ usando una promedio ponderado dado por

$$\boldsymbol{\mu}^{(\boldsymbol{t+1})} \leftarrow \sum_{i=1}^{n} w_i \boldsymbol{x_i}, \tag{1.17}$$

donde w_i son valores fijos y escogidos de tal manera que proporcionen mayor contribución a los puntos con menor valor al evaluarlos en f. Esto permite mover la media $\boldsymbol{\mu}$ en una dirección favorable.

Seguidamente, se necesita actualizar σ para expandir o reducir la hiperelipse en la siguiente iteración. Por este motivo, la línea 5 controla este valor mediante las ecuaciones

$$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{c_\sigma}{d_\sigma} \underbrace{\left(\frac{||\boldsymbol{p_\sigma}||}{\mathbb{E}||\mathcal{N}(0, \mathbf{I})||} - 1\right)}_{\text{evolution path comparison}}\right), \tag{1.18}$$

$$\mathbb{E}||\mathcal{N}(0, \mathbf{I})|| = \sqrt{2}\left(\frac{\Gamma\left(\frac{p+1}{2}\right)}{\Gamma\left(\frac{p}{2}\right)}\right), \tag{1.19}$$

donde $\boldsymbol{p_\sigma}$ es una variable que acumula los pasos llevados, $c_\sigma \in [0, 1]$ es una variable que determina el tiempo acumulado para p_σ y $d_\sigma \approx 1$ es un parámetro que determina el ratio de posibilidad de cambio de $\sigma^{(t+1)}$. La principal parte de la Ecuación 1.18 es el

término *evolution path comparison*, aquí se compara el tamaño de p_σ con su tamaño esperado bajo selección aleatoria. De esta comparación podemos controlar si el valor de σ debe incrementarse, disminuirse o permanecer igual.

Finalmente, en la línea 6 cambiamos $\boldsymbol{C}$ a una dirección favorable usando lo siguiente:

$$\boldsymbol{C}^{(\boldsymbol{t+1})} \leftarrow \overbrace{\left(1 - c_1 c_c (1 - h_\sigma)(2 - c_c) - c_1 - c_\mu\right)\boldsymbol{C}^{(\boldsymbol{t})}}^{\text{cumulative update}} + \underbrace{c_1 \boldsymbol{p_C} \boldsymbol{p_C}^T}_{\text{rank-one update}} + \underbrace{c_\mu \sum_{i=1}^{n} w_i' \boldsymbol{\delta}^{(\boldsymbol{i})} \left(\boldsymbol{\delta}^{(\boldsymbol{i})}\right)^T}_{\text{rank-}\mu\text{ update}}, \quad (1.20)$$

donde $c_\mu \leq 1$ es el radio de aprendizaje para el término *rank-μ update*, $c_1 \leq 1 - c_\mu$ es el radio de aprendizaje para el término *rank-one update*, $c_c \in [0, 1]$ es el radio de aprendizaje para el término *cumulative update*, h_σ es la evaluación bajo la función unitaria usado para actualizar apropiadamente el camino evolutivo, $\boldsymbol{p_C}$ es un vector acumulativo usado para actualizar la matriz de covarianza, w_i' son los coeficientes de ponderación modificados y $\boldsymbol{\delta}^{(\boldsymbol{i})}$ son las desviaciones seleccionadas.

En la Ecuación 1.20, el primer término (*cumulative update*) mantiene información de la anterior matriz de convarianza. El segundo término (*rank-one update*) permite expandir la distribución en una dirección favorable. El tercer término (*rank-μ update*) incrementa la búsqueda en espacios donde es probable encontrar buenas soluciones. La combinación de estos tres términos actualiza $\boldsymbol{C}$ de tal manera que mueva la hiperelipse en una dirección favorable. Para una descripción más detallada del algoritmo, revisar los trabajos de Hansen (2016) y Kochenderfer y Wheeler (2019).

1.5.4 *Gradient Descent* (GD)

Este algoritmo comienza desde un punto inicial, en nuestro caso la matriz $\boldsymbol{P}$. Luego, usa la derivada en ese punto para guiar su búsqueda iterativamente, esto lo realiza mediante la siguiente ecuación:

$$\boldsymbol{P^{(t+1)}} \leftarrow \boldsymbol{P^{(t)}} - \gamma \nabla \boldsymbol{P^{(t)}}, \tag{1.21}$$

donde t representa la iteración actual y γ (conocido como el ratio de aprendizaje) determina la magnitud de como actualizar $\boldsymbol{P^{(t)}}$. Particularmente, un enfoque razonable para buscar convergencia a un mínimo local es actualizar γ en cada iteración mediante la ecuación

$$\gamma^{(t+1)} = \frac{|(\boldsymbol{P^{(t)}} - \boldsymbol{P^{(t-1)}})^T (\nabla f(\boldsymbol{P^{(t)}}) - \nabla f(\boldsymbol{P^{(t-1)}}))|}{||\nabla f(\boldsymbol{P^{(t)}}) - \nabla f(\boldsymbol{P^{(t-1)}})||^2}. \tag{1.22}$$

Un detalle importante a señalar es que por simplicidad en la notación, la Ecuación 1.22 se describió de esta manera; sin embargo, en realidad debemos trabajar con la matriz $\boldsymbol{P}$ después de ser linealizada para que las operaciones de esta ecuación estén bien definidas. Mayores detalles del algoritmo se pueden encontrar en Demidova y Gorchakov (2020).

1.5.5 *Method of Moving asymptotes* (MMA)

Method of moving asymptotes (MMA) es un algoritmo de optimización local basado en la gradiente, es decir, de primer orden. La idea del algoritmo es comenzar en un punto $\boldsymbol{P}$ y formar una aproximación de f alrededor de $\boldsymbol{P}$ utilizando: (i) la gradiente de f, (ii) los límites de los valores de P y (iii) una función de penalidad cuadrática. Lo interesante de esta aproximación es que es convexa y separable, lo cual permite resolverla de

forma sencilla. Así, el algoritmo comienza calculando la aproximación, luego resuelve la optimización utilizando la aproximación. De este modo, surgen dos posibles escenarios: (i) la aproximación encuentra un buen elemento que se usa como punto de inicio para repetir el procedimiento o (ii) se repite el proceso limitando la contribución de la función de penalidad.

En realidad en el presente trabajo usaremos una versión mejorada del algoritmo conocida como *Conservative Convex Separable Approximation* (CCSA) debido a su popularidad en optimización topológica, pero nos seguiremos refiriendo al algoritmo como MMA por convención. Esta variante tiene la particularidad de lograr converger a un mínimo local independientemente del punto inicial. Los detalles se pueden revisar en Svanberg (2002).

1.5.6 *Limited-memory Broyden–Fletcher–Goldfarb–Shanno with boundaries* (L-BFGS-B)

Limited-memory Broyden–Fletcher–Goldfarb–Shanno with boundaries (L-BFGS-B) es un algoritmo de optimización que se caracteriza por estimar la inversa de la matriz hessiana utilizando una cantidad de memoria de orden lineal. El algoritmo comienza con un punto inicial $\boldsymbol{P}$. Luego, internamente mantiene el historial de las últimas evaluaciones de la función f y ∇f para determinar una dirección para actualizar $\boldsymbol{P}$. Esto es posible precondicionando la gradiente con información de curvatura obtenida por la aproximación de la matriz hessiana. El lector interesado puede revisar los detalles del algoritmo en Liu y Nocedal (1989).

En el presente capítulo se ha descrito la teoría necesaria para optimizar el diseño de un un *bend* y WDM. Esto ha implicado la explicación de la parametrización basada en píxeles, el uso de SPINS-B y transformaciones utilizadas en la optimización topológica robusta. Además, se ha desarrollado la explicación de tres algoritmos que no necesitan

calcular la gradiente: (i) GA, (ii) PSO y (iii) CMA-ES. Asimismo, se describió tres algoritmos de primer orden (que usan la gradiente): (i) GD, (ii) MMA y (iii) L-BFGS-B. Los conceptos presentados serán de utilidad para entender la propuesta de esta tesis descrita en el Capítulo 3.

Capítulo 2

Trabajos Relacionados

En el presente capítulo comenzaremos discutiendo sobre un trabajo que optimiza un *splitter*. Seguidamente, identificaremos dos inconvenientes con este trabajo e iremos mostrando como otras investigaciones han afrontado estos desafíos.

En Prosopio-Galarza *et al.* (2019) se optimizó un *splitter* con guías de onda fijas de $0.5\mu m$, donde las guías de onda de salida son separadas por $0.2\mu m$ y todas estas son unidas con una región rectangular de diseño de $2\mu m \times 1.5\mu m$. Con esta geometría se simulan distintos diseños dividiendo la región rectangular en ($z = 13$) segmentos uniformemente separados. Cada segmento puede variar su altura dentro del rectángulo, estos se centran de forma vertical y se van uniendo sus extremos. La representación de esta idea la podemos observar en la Figura 2.1 con $z = 5$ segmentos.

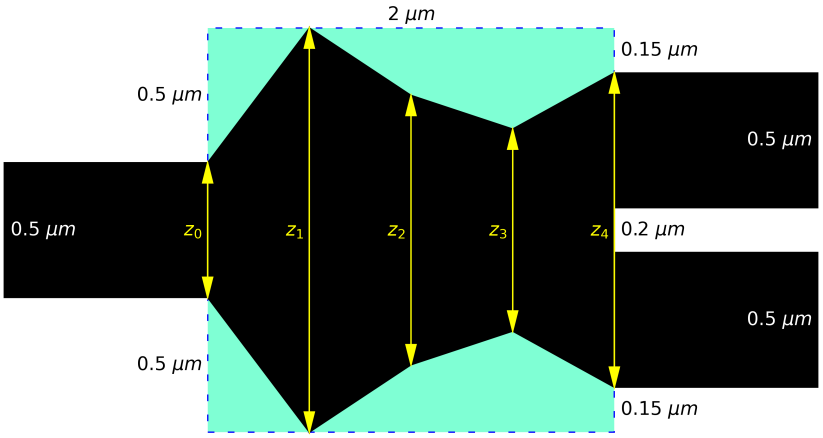


FIGURA 2.1: Diseño de un *splitter* basado en (Prosopio-Galarza *et al.*, 2019) utilizando $z = 5$ segmentos.

Como función objetivo se establece maximizar la transmitancia en la guía de onda superior trabajando con una longitud de onda de $1550nm$. Los mejores resultados son obtenidos al usar PSO como algoritmo de optimización.

Es destacable que al usar esta parametrización se puede limitar las alturas de los segmentos para asegurar obtener ángulos agudos, los cuales son los más adecuados como regla práctica de diseño (Lukas Chrostowski, 2010). Sin embargo, hay dos inconvenientes con este trabajo:

- **Inconveniente 1:** La parametrización utilizada descarta la posibilidad de diseños menos intuitivos (por ejemplo, con agujeros) que podrían ocupar menor área y mantener una buena transmitancia.
- **Inconveniente 2:** Las optimizaciones solo se repitieron una vez con apenas 30 iteraciones y una población de 14 individuos.

A continuación, vamos a desarrollar en más detalle como otros trabajos han buscado solucionar estos dos inconvenientes.

2.1 Inconveniente 1: La Parametrización

Como se detalla en Molesky *et al.* (2018), para permitir diseños con mayor grado de libertad se suele trabajar con dos enfoques: (i) parametrización basada en conjuntos de nivel y (ii) parametrización basada en píxeles.

La parametrización basada en conjuntos de nivel permite describir geometrías poco intuitivas, con elevada transmitancia y sin necesidad de regiones grises. Piggott *et al.* (2015) utiliza esta parametrización para optimizar un WDM con guías de onda de $0.5\mu m$ y una región de diseño de $2.8\mu m \times 2.8\mu m$ trabajando a $1300nm$ y $1550nm$. Comienza con un diseño inicial aleatorio, luego realiza la optimización en tres etapas consecutivas:

(i) optimiza sin restricciones con una parametrización basada en píxeles (pudiendo producir regiones grises), (ii) introduce una parametrización basada en conjuntos de nivel, (iii) busca optimizar en distintas longitudes de onda con el fin de buscar un diseño robusto. Este enfoque ha dado buenos resultados incluso al fabricarse. Además, en Piggott *et al.* (2017) se desarrolla un algoritmo para tratar de imponer un mínimo radio de curvatura a los diseños y asegurar mayor coherencia entre las simulaciones y los diseños fabricados.

De manera similar, Su *et al.* (2020) obtiene transmitancias alrededor de $90\,\%$ al optimizar un WDM trabajando a $1400nm$ y $1550nm$. Sin embargo, su proceso de optimización consiste en solo dos etapas: (i) optimización continua y (ii) optimización discreta. Lo interesante de su propuesta es que en la optimización continua trabaja con subetapas donde aplica una transformación sigmoidal (similar a la Ecuación 1.10) para ir discretizando los diseños. Luego, en la optimización discreta aplica una parametrización basada en conjuntos de nivel para eliminar regiones grises e imponer un mínimo radio de curvatura de $100nm$.

Por otro lado, la parametrización basada en píxeles también ha obtenido buenos resultados en la optimización topológica. Como se describe en Lazarov *et al.* (2016), generalmente se utiliza las transformaciones descritas en la Sección 1.4: filtro por densidad y proyección. Particularmente, utilizando estas mismas transformaciones se logra simular dos posibles errores de fabricación: dilatación y erosión. A modo de tutorial, Christiansen y Sigmund (2021b) optimiza un WDM con guías de onda de $0.299\mu m$ y una región de diseño de $2\mu m \times 2\mu m$ trabajando a $1300nm$ y $1550nm$. Utilizando parametrización basada en píxeles, realiza un proceso similar a la optimización continua de Su *et al.* (2020); pero, en vez de aplicar una función sigmoidal para discretizar el diseño, aplica el filtro por densidad para tratar de imponer un mínimo radio de curvatura y después la proyección para discretizar los diseños. Asimismo, define su función objetivo de tal modo que asegura diseños robustos ante dilatación o erosión.

De manera similar Hammond *et al.* (2020) trabaja con optimización topológica robusta para optimizar un *splitter* en una región de diseño de $3\mu m \times 3\mu m$. En este trabajo se realiza la optimización imponiendo distintos radios de curvatura. De sus resultados se puede observar que a menor radio de curvatura se logra definir más detalles en los diseños y la optimización logra mejores resultados. Posteriormente, Hammond *et al.* (2021) fabrica diseños obtenidos con esta idea y muestra buenos resultados. Pero, detallan la presencia de pequeñas variaciones en las simulaciones y datos experimentales posiblemente causada por regiones aisladas en el diseño.

2.2 Inconveniente 2: La Optimización

En Malheiros-Silveira y Delalibera (2020) se compararon dos algoritmos en la optimización de un dispositivo. A diferencia de Prosopio-Galarza *et al.* (2019), la comparación no se realizó en base a la cantidad de iteraciones realizadas por cada algoritmo, en cambio se hizo de acuerdo a la cantidad de simulaciones realizadas (≈ 2000), es decir, la cantidad de veces que se evaluó la FOM. Esta estrategia es más adecuada para comparar algoritmos con distintas características de una manera más justa.

En otros trabajos que se centran en comparar algoritmos para optimizar un mismo dispositivo se sigue la misma idea para la comparación (Gregory *et al.*, 2015, Schneider *et al.*, 2019); sin embargo, no parece haber un consenso sobre algún algoritmo que funcione bien para optimizar cualquier dispositivo.

En general, PSO y GA han sido extensamente usados en el área de acuerdo de acuerdo a reseñas como las de Elsawy *et al.* (2020) y Campbell *et al.* (2019). Tal y como es señalado en estos trabajos, el desempeño de ambos algoritmos es sensible a los parámetros escogidos. Este es un gran inconveniente debido a que escoger parámetros adecuados

puede consumir mucho tiempo y esto se debe hacer independientemente para cada dispositivo. Con el propósito de superar esta dificultad, distintos trabajos están optando por usar algoritmos que no necesiten configurar parámetros internos.

Bajo este enfoque, en Gregory *et al.* (2015) se resaltó el buen desempeño que pude obtener el algoritmo CMA-ES en la optimización de ciertos dispositivos, llegando a superar al PSO. De manera similar, en Schneider *et al.* (2019) se realizó un trabajo muy completo comparando distintos algoritmos llegando a resultados donde la optimización bayesiana mostró los resultados más prometedores, pero esta comparación fue para un bajo número de parámetros (< 15). Y, aunque no detalla la razón, en Su *et al.* (2020) se empleó el algoritmo L-BFGS-B en la optimización de un *bend* y WDM llegando a obtener resultados destacados.

Un aspecto importante a resaltar de estos últimos tres trabajos es que al comparar distintos algoritmos para optimizar dispositivos fotónicos necesitamos contar con (i) un elevado número de simulaciones y (ii) distintas ejecuciones que comiencen con diferentes puntos iniciales.

Como se ha discutido en este capítulo, la parametrización de nuestros dispositivos usando un bajo número de parámetros puede ir condicionando nuestros resultados, ante ellos dos posibles soluciones son usar (i) parametrización basada en conjuntos de nivel o (ii) parametrización basada en píxeles. Esto supone nuevos desafíos, mas ya existen estrategias para afrontarlos e incluso para incluir restricciones de fabricación a ambas parametrizaciones utilizando distintas estrategias de optimización. Por otro lado, aunque no se mencionó explícitamente, en el área de fotónica no parece haber muchas investigaciones que comparen distintos algoritmos cuando trabajamos con un elevado número de parámetros para representar nuestros dispositivos, en especial si los algoritmos no son PSO o GA.

Capítulo 3

Metodología

En el presente capítulo se describe la metodología a seguir en esta tesis. Primero, se describe de forma general los seis pasos que se trabajaran. Luego, se detalla en una sección completa cada uno de estos. Finalmente, se brindan los alcances y limitaciones de la propuesta.

Como se muestra en la Figura 3.1, en esta investigación se siguieron los siguientes seis pasos: (i) preparación de simulación, (ii) preparación de optimización, (iii) optimización continua, (iv) optimización discreta, (v) optimización de fabricación y (vi) preparación para fabricación.

Estos pasos se siguieron para optimizar tanto un *bend* como un WDM. Una vez preparada la simulación, la etapa de optimización (continua, discreta y de fabricación) se realizó tres veces por cada algoritmo. La primera ejecución usa un valor de semilla de 128, la segunda de 256 y la tercera de 512. De esta manera se aseguró iniciar con diseños aleatorios y mantener los resultados reproducibles.

Es importante señalar que los resultados de la optimización continua se usan como punto de inicio para la optimización discreta. Asimismo, los resultados de la optimización discreta se utilizan como entrada para la optimización de fabricación. Este proceso tiene como fin el mantener un buen resultado (Yang y Fan, 2017).

En particular, el valor de β, presente en la Ecuación 1.10, representa el factor discretizador de nuestros diseños. Este valor se va incrementando en la optimización discreta

y de fabricación como se muestra en la Figura 3.1. Finalmente, tras haber conseguido diseños optimizados se realizó un posprocesamiento, análisis de los diseños más optimos y conversión al formato GDSII para dejar los diseños listos para fabricación.

En las siguientes subsecciones se explica en detalle cada una de los pasos de la metodología seguida en esta tesis.

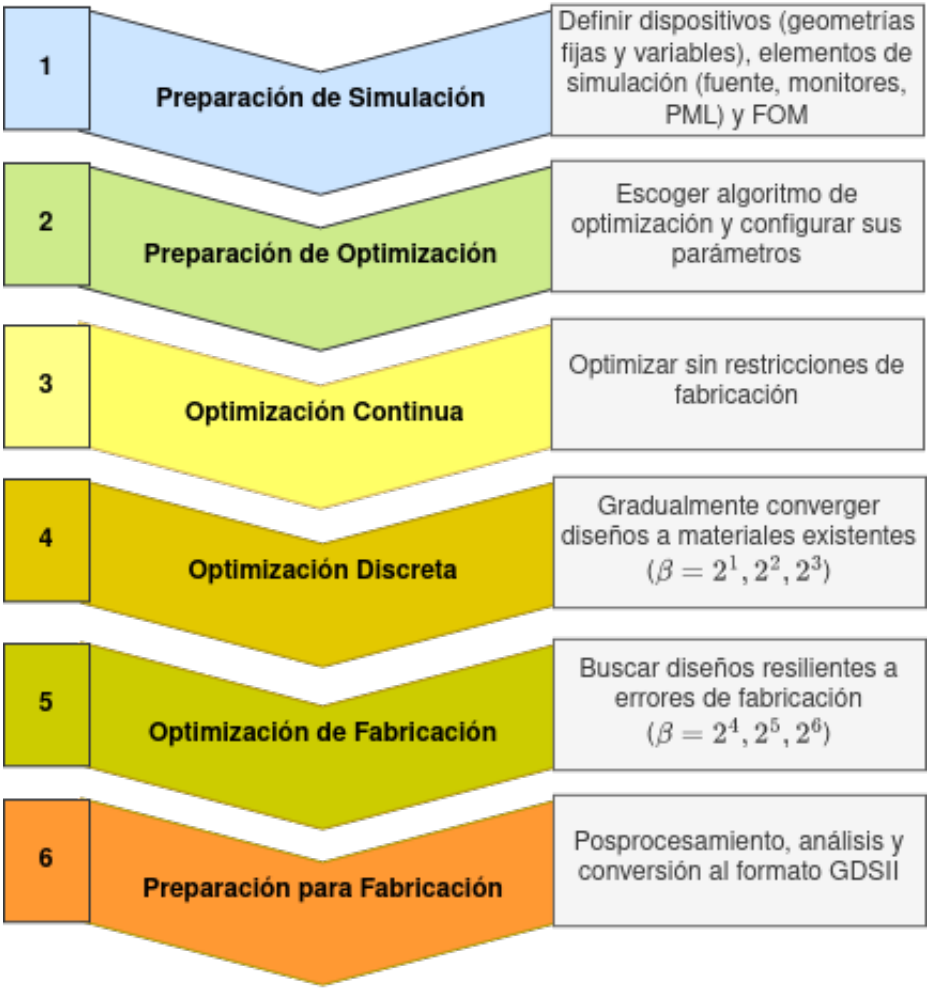


FIGURA 3.1: Metodología del trabajo de investigación

3.1 Preparación de Simulación

El *bend* y WDM se parametrizaron usando la parametrización basada en píxeles descrita en la Sección 1.2. La implementación se realizó en SPINS. La descripción detallada para los dos dispositivos de estudio se presentan en las siguientes dos subsecciones.

3.1.1 *Bend*

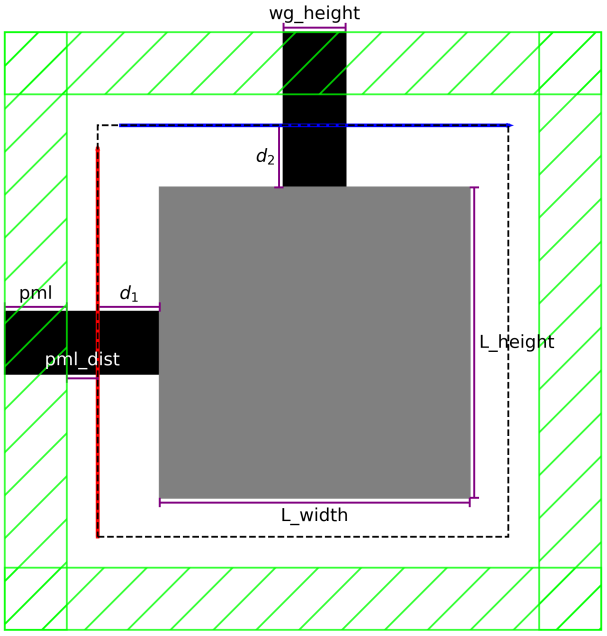


FIGURA 3.2: Parámetros del diseño del *bend* a optimizar.

En la Figura 3.2 se muestra el diseño y parámetros del *bend* a utilizar. Los rectángulo negros representan las guías de onda, estos tienen las mismas dimensiones aunque en distinta orientación. La región gris representa la región de diseño. La recta roja simula la fuente y la recta azul el monitor, su extensión está simbolizada por las variables *source* y *monitor*, respectivamente. Además, la profundidad de estos dos elementos es la misma,

valor denotado como *z_length*. Por otro lado, la región punteada se utiliza como referencia para definir el PML (región verde), del cual dista un valor definido como *pml_dist*. La profundidad de toda la geometría es especificada por la variable *depth*.

Cabe destacar que los parámetros fueron escogidos inspirados por el trabajo de Su *et al.* (2020). De este modo, la región de diseño se dividió en rectángulos de $16nm \times 16nm$, obteniendo así una matriz de 125×125. Notemos que con esta configuración, en la Ecuación 1.5 tenemos $psize_x = psize_y = 16nm$. El valor de los demás parámetros del diseño se presenta en la Tabla 3.1.

Parámetro	Valor (nm)
wg_height	400
pml	300
pml_dist	200
d_1	400
d_2	400
L_width	2000
L_height	2000
source	2500
monitor	2500
z_length	1000
depth	220

TABLA 3.1: Parámetros usados en el diseño del *bend* a optimizar.

3.1.2 WDM

En la Figura 3.3 se muestra el diseño y parámetros del WDM a utilizar. Los rectángulo negros representan las guías de onda, estos tienen las mismas dimensiones. La región gris representa la región de diseño. La recta roja simula la fuente y su extensión se simboliza por la variable *source*. Las rectas azul y marrón describen los monitores cuya extensión está definida por las variables *monitor*. Además, la profundidad de la fuente y

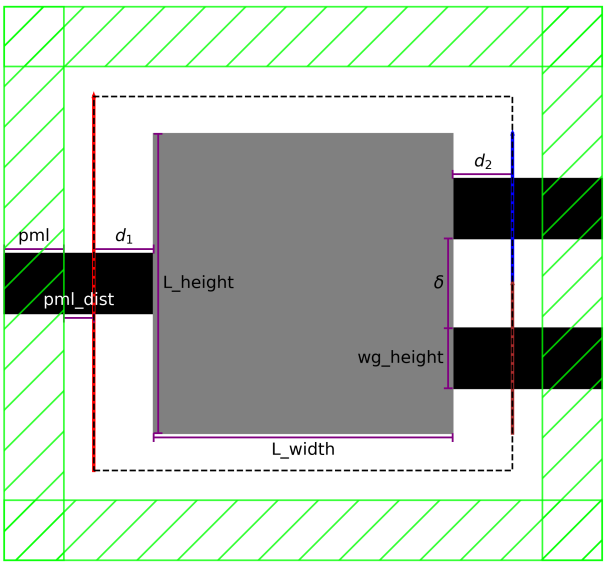


FIGURA 3.3: Parámetros del diseño del WDM a optimizar.

monitores es la misma, denotado como *z_length*. Por otro lado, la región punteada se utiliza como referencia para definir el PML (región verde), del cual dista un valor definido como *pml_dist*. La profundidad de toda la geometría es especificada por la variable *depth*.

En este caso, los parámetros fueron escogidos inspirados por el trabajo de Christiansen y Sigmund (2021b). Del mismo modo como se realizó para el *bend*, la región de diseño se dividió en rectángulos de $16nm \times 16nm$, obteniendo así una matriz de 125×125. El valor de los demás parámetros se presenta en la Tabla 3.2.

3.2 Preparación de Optimización

Con lo descrito en la anterior sección ya podemos evaluar distintos diseños. Ahora, utilizando como función objetivo la Ecuación 1.2 para el *bend* y la Ecuación 1.3 para el WDM, estamos ante un problema de optimización. Para resolverlo, la propuesta inicial era utilizar estos cinco algoritmos: (i) GA, (ii) PSO, (iii) CMA-ES, (iv) L-BFGS-B, (v) MMA.

Parámetro	Valor (nm)
wg_height	400
pml	300
pml_dist	200
d_1	400
d_2	400
L_width	2000
L_height	2000
δ	600
source	2500
monitor	1000
z_length	1000
depth	220

TABLA 3.2: Parámetros usados en el diseño del WDM a optimizar.

La elección de estos fue por diversos motivos. GA y PSO se escogieron por ser populares en el área (Elsawy *et al.*, 2020, Molesky *et al.*, 2018, Prosopio-Galarza *et al.*, 2019). CMA-ES por haber tenido un buen desempeño en Gregory *et al.* (2015) y las buenas referencias brindadas en Campbell *et al.* (2019). L-BFGS-B por haber obtenido buenos resultados en trabajos como Su *et al.* (2020). MMA por haber sido recomendado como un buen candidato para la optimización topológica (Christiansen y Sigmund, 2021b).

Por trabajos como los de Su *et al.* (2020) y Christiansen y Sigmund (2021b) ya nos podíamos anticipar que al usar L-BFGS-B o MMA se obtendrían buenos resultados en parte gracias a usar la gradiente para guiar la búsqueda. Sin embargo, por la cantidad de variables utilizadas (125×125) no podíamos asegurar lo mismo con GA, PSO y CMA.

Debido a ello, primero se realizó un experimento preliminar para evaluar el desempeño de estos algoritmos al realizar como máximo 1200 evaluaciones en la optimización del *bend* propuesto, estos resultados son presentados en la Figura 3.4. Como podemos observar, especialmente del CMA-ES, hay una ligera tendencia de mejora. Por otro lado, por los resultados mostrados en Christiansen y Sigmund (2021a) uno esperaría que algoritmos

como GA lograran obtener resultados similares a los obtenidos por algún algoritmo basado en la gradiente (e.g. L-BFGS-B y MMA); sin embargo, esto podría necesitar evaluar el FOM un número muy elevado de veces (entre 10^4 y 10^5).

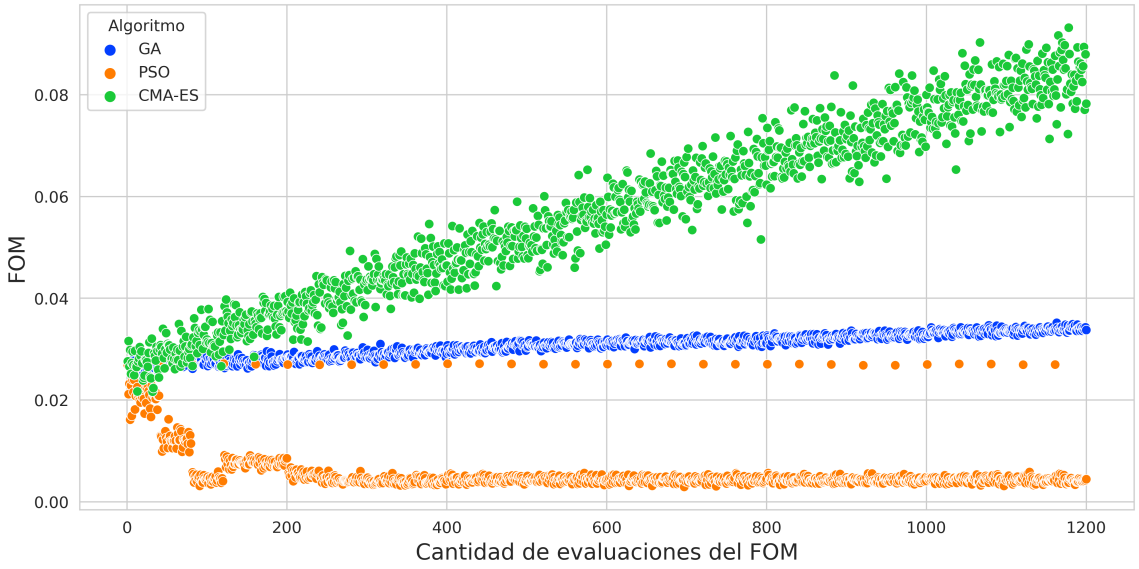


FIGURA 3.4: Comparación del desempeño de GA, PSO y CMA-ES para optimizar el *bend* propuesto.

Algoritmo	Parámetro	Valor (nm)
G-GA	population_size	40
G-GA	n_selected_parents	10
G-GA	GA_range	0.1
G-GA	prob_mutation	0.1
G-PSO	population_size	40
G-PSO	PSO_range	0.1
G-PSO	ω	0.5
G-PSO	c_1	0.5
G-PSO	c_2	0.5
G-CMA-ES	population_size	40
G-CMA-ES	σ	0.3

TABLA 3.3: Parámetros usados por los algoritmos de optimización.

De este modo, en esta tesis se optó por comparar solamente algoritmos de primer orden, es decir, que utilicen el cómputo de la gradiente en sus rutinas. Así, se seguirá evaluando los algoritmos GA, PSO y CMA-ES pero ahora en sus versiones que usan la

gradiente. Para poder comparar el desempeño de estos se configuró la máxima cantidad de veces que podían evaluar la función objetivo. Los parámetros específicos con los que se ejecutaron cada algoritmo se detallan en la Tabla 3.3 siguiendo la nomenclatura presentada en la Sección 1.5.

Es resumen, usando simulaciones en 3D FDFD con SPINS-B, evaluamos estos cinco algoritmos:

1. **G-PSO**

   Llamemos así a la versión de PSO (Subsección 1.5.2) que usa la gradiente. Basándonos en el trabajo de Demidova y Gorchakov (2020), después de cada iteración del Algoritmo 2, aplicamos el algoritmo de GD (Subsección 1.5.4) para actualizar la mejor solución global encontrada por el algoritmo PSO. Esta aplicación de GD se realiza por 10 iteraciones. La idea detrás de esta variente es asegurar que la mejor solución global alcance un buen resultado para lograr guiar a las demás partículas (manteniendo así el enfoque de un algoritmo de optimización global).

   Los experimentos con este algoritmo se desarrollaron con una implementación propia.

2. **G-GA**

   Similar al G-PSO, después de cada iteración del Algoritmo 1, se escoge dos elementos aleatorios para actualizarlos aplicándoles GD por 10 iteraciones. Se escoge dos elementos y no solo uno para asegurar que los resultados de aplicar GD se compartan rápidamente con los demás individuos en pocas iteraciones.

   Los experimentos con este algoritmo se desarrollaron con una implementación propia.

3. **G-CMA-ES**

En esta variante, al final de cada iteración del Algoritmo 3 se calcula la gradiente de la media obtenida, luego este valor se utiliza para actualizar la hiperelipse en la siguiente iteración. Los detalles precisos se pueden encontrar en Akimoto y Hansen (2021).

Los experimentos con este algoritmo se desarrollaron usando el paquete de Python pycma (Hansen *et al.*, 2019).

4. **MMA** (Subsección 1.5.5)

   No se realizaron variantes en este algoritmo. Los experimentos se desarrollaron usando el paquete de Python NLOpt (Johnson, 2022).

5. **L-BFGS-B** (Subsección 1.5.6)

   No se realizaron variantes en este algoritmo. Los experimentos se desarrollaron usando el paquete de Python SciPy (Virtanen *et al.*, 2020).

A continuación, se detalla la estrategia de optimización seguida en este trabajo. Esta propuesta esta inspirada (principalmente) en el trabajo de Su *et al.* (2020) y Christiansen y Sigmund (2021b). En cada una de estas etapas se limitó la cantidad de veces que se podía evaluar f_{obj} para su posterior comparación. Además, para G-GA y G-PSO se agregó una condición para terminar el algoritmo en caso de no encontrar mejoras mayores a 10^{-9} en las últimas 5 iteraciones, para el L-BFGS-B se usó valores por defecto y para MMA se estableció un toleracia relativa de 10^{-9}.

3.3 Optimización Continua

En esta etapa se configuró la máxima cantidad de evaluaciones a 2000. Luego, se ejecutaron los cinco algoritmos para optimizar el *bend* y WDM. Cada algoritmo se ejecutó 3 veces, la primera con un valor de semilla de 128, la segunda de 256 y la tercera de 512. La idea de esta etapa es optimizar los diseños sin imponer ninguna restricción. Es decir,

directamente usamos la parametrización $\boldsymbol{P}$ para evaluar la permitividad ε (Ecuación 1.4), lo cual lo podemos representar como $(\boldsymbol{P} \rightsquigarrow \varepsilon)$.

3.4 Optimización Discreta

Los objetivos de esta etapa son dos: (i) obtener diseños con un mínimo radio de curvatura r_f e (ii) ir eliminando las regiones grises. Para el primer objetivo se aplicó la Ecuación 1.7 a la parametrización $\boldsymbol{P}$ usando un radio de curvatura $r_f = 80nm$. Como $80nm$ es igual a 5 veces el tamaño de los píxeles configurados en la Sección 3.1, solamente fue necesario explorar una submatriz de 11×11 alrededor de cada elemento para el cálculo de esta ecuación. Para el segundo objetivo se aplicó la Ecuación 1.10 al resultado conseguido por el anterior filtro. Este proceso lo podemos representar como $(\boldsymbol{P} \rightsquigarrow \widetilde{\boldsymbol{P}} \rightsquigarrow \widetilde{\widetilde{\boldsymbol{P}}} \rightsquigarrow \varepsilon)$.

Para la optimización se utilizó los resultados de la optimización continua como punto inicial. Para ello se realizó una primera optimización configurando la máxima cantidad de evaluaciones en 800 y aplicando a cada diseño los filtros descritos anteriormente con $\beta = 2^1$ y $\eta = 0.5$ para la Ecuación 1.10. El resultado de esta optimización se usa como punto inicial para repetir este proceso, pero ahora con $\beta = 2^2$. Finalmente, se repite una vez más el procedimiento con $\beta = 2^3$.

3.5 Optimización de Fabricación

Por último, se busca imponer a los diseños restricciones de fabricación y robustez a posibles errores de fabricación.

Tomando como referencia el trabajo de Hammond *et al.* (2020) para asegurar un buen desempeño pese a los errores de fabricación, por cada parametrización $\boldsymbol{P}$ se calculó:

- $\widetilde{\widetilde{\boldsymbol{P}}}_d$ que representa el diseño como si el dispositivo se hubiera dilatado.
- $\widetilde{\widetilde{\boldsymbol{P}}}_i$ que representa el diseño normal.
- $\widetilde{\widetilde{\boldsymbol{P}}}_e$ que representa el diseño como si el dispositivo se hubiera expandido.

Estos tres elementos se calcularon siguiendo los mismos filtros descritos en la anterior sección ($\boldsymbol{P} \rightsquigarrow \widetilde{\boldsymbol{P}} \rightsquigarrow \widetilde{\widetilde{\boldsymbol{P}}} \rightsquigarrow \varepsilon$). Pero, para el diseño dilatado se usó $\eta_d = 0.45$, para el diseño normal $\eta_i = 0.5$ y para el diseño expandido $\eta_e = 0.55$.

Luego, se definió una nueva función objetivo mediante la siguiente ecuación:

$$F_{obj} = max(min(f_{obj}(\widetilde{\widetilde{\boldsymbol{P}}}_d), f_{obj}(\widetilde{\widetilde{\boldsymbol{P}}}_i), f_{obj}(\widetilde{\widetilde{\boldsymbol{P}}}_e)) \tag{3.1}$$

Así, considerando el peor escenario ante dos posibles errores de fabricación (expansión o contracción), se busca el diseño más resiliente posible. Para optimizar esta nueva función objetivo se realizó el mismo proceso de la optimización discreta, pero ahora con valores $\beta = 2^4, 2^5, 2^6$ y limitando a 400 evaluaciones de F_{obj} por cada β. De esta manera, en paralelo, seguimos con el proceso de discretizar nuestros resultados.

3.6 Preparación para Fabricación

Finalmente, se seleccionó el diseño más optimo obtenido para un *bend* y WDM. Luego, se realizó un analisis del funcionamiento de estos dispositivos a distintas longitudes de onda y se evaluó su desempeño al eliminar regiones aisladas. Posteriormente, usando la permitividad asociada a sus parametrization se obtuvo la geometría del dispositivo. Esta geometría se aproximó mediante polígonos usando el paquete *mathplotlib* de Python para representar el diseño en formato GDSII y dejarlo listo para fabricación.

3.7 Alcances y Limitaciones

El presente trabajo solo se realizó mediante simulaciones computacionales, no se llegó a la parte de fabricación por temas de tiempo y dinero.

En este capítulo hemos explicado los seis pasos seguidos en esta tesis. Primero, comenzamos describiendo la configuración de nuestros dispositivos a optimizar. Segundo, se detalló los cinco algoritmos a utilizar, sus parámetros y justificación de elección. Luego, se desarrolló la estrategia de optimización a seguir: optimización continua, discreta y de fabricación. Finalmente, se describió la etapa final realizada con los mejores diseños obtenidos.

REFERENCIAS BIBLIOGRÁFICAS

Akimoto, Y. and Hansen, N. (2021). Cma-es and advanced adaptation mechanisms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, page 636–663, New York, NY, USA. Association for Computing Machinery.

Angeris, G., Vučković, J., and Boyd, S. (2021). Heuristic methods and performance bounds for photonic design. *Opt. Express*, 29(2):2827–2854.

Campbell, S. D., Sell, D., Jenkins, R. P., Whiting, E. B., Fan, J. A., and Werner, D. H. (2019). Review of numerical optimization techniques for meta-device design [invited]. *Opt. Mater. Express*, 9(4):1842–1863.

Christiansen, R. E. and Sigmund, O. (2021a). Compact 200 line matlab code for inverse design in photonics by topology optimization: tutorial. *J. Opt. Soc. Am. B*, 38(2):510–520.

Christiansen, R. E. and Sigmund, O. (2021b). Inverse design in photonics by topology optimization: tutorial. *J. Opt. Soc. Am. B*, 38(2):496–509.

Demidova, L. A. and Gorchakov, A. V. (2020). Research and study of the hybrid algorithms based on the collective behavior of fish schools and classical optimization methods. *Algorithms*, 13(4).

Elsawy, M. M., Lanteri, S., Duvigneau, R., Fan, J. A., and Genevet, P. (2020). Numerical Optimization Methods for Metasurfaces. *Laser and Photonics Reviews*, 14(10):1–17.

Gregory, M. D., Martin, S. V., and Werner, D. H. (2015). Improved Electromagnetics Optimization. *IEEE Antennas and Propagation Magazine*, 57(june):48–59.

Hammond, A. M., Oskooi, A., Johnson, S. G., and Ralph, S. E. (2020). Robust topology optimization of foundry-manufacturable photonic devices: An open-source fdtd toolbox. In *Frontiers in Optics / Laser Science*, page FTh1C.4. Optical Society of America.

Hammond, A. M., Slaby, J., Saha, G., and Ralph, S. E. (2021). Robust topology optimization for foundry-photonics inverse design: Examining compact and arbitrary power splitters. In *2021 European Conference on Optical Communication (ECOC)*, pages 1–4.

Hansen, N. (2016). The CMA Evolution Strategy: A Tutorial.

Hansen, N., Akimoto, Y., and Baudis, P. (2019). CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634.

Johnson, S. G. (2022). The nlopt nonlinear-optimization package.

Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for Optimization*. The MIT Press.

Lazarov, B. S., Wang, F., and Sigmund, O. (2016). Length scale and manufacturability in density-based topology optimization. *Archive of Applied Mechanics*, 86(1-2):189–218.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528.

Lukas Chrostowski (2010). *Silicon Photonics Design: From Device to System*.

Malheiros-Silveira, G. N. and Delalibera, F. G. (2020). Inverse design of photonic structures using an artificial bee colony algorithm. *Applied Optics*, 59(13):4171.

Molesky, S., Lin, Z., Piggott, A. Y., Jin, W., Vucković, J., and Rodriguez, A. W. (2018). Inverse design in nanophotonics. *Nature Photonics*, 12(11):659–670.

Oskooi, A. F., Roundy, D., Ibanescu, M., Bermel, P., Joannopoulos, J. D., and Johnson, S. G. (2010). Meep: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications*, 181(3):687–702.

Piggott, A. Y., Lu, J., Lagoudakis, K. G., Petykiewicz, J., Babinec, T. M., and Vučković, J. (2015). Inverse design and demonstration of a compact and broadband on-chip wavelength demultiplexer. *Nature Photonics*, 9(6):374–377.

Piggott, A. Y., Petykiewicz, J., Su, L., and Vučković, J. (2017). Fabrication-constrained nanophotonic inverse design. *Scientific Reports*, 7(1):1–7.

Prosopio-Galarza, R., De La Cruz-Coronado, J., Hernandez-Figueroa, H. E., and Rubio-Noriega, R. (2019). Comparison between optimization techniques for Y-junction devices in SOI substrates. *Proceedings of the 2019 IEEE 26th International Conference on Electronics, Electrical Engineering and Computing, INTERCON 2019*, pages 1–4.

Schneider, P. I., Garcia Santiago, X., Soltwisch, V., Hammerschmidt, M., Burger, S., and Rockstuhl, C. (2019). Benchmarking Five Global Optimization Approaches for Nano-optical Shape Optimization and Parameter Reconstruction. *ACS Photonics*, 6(11):2726–2733.

Su, L., Vercruysse, D., Skarda, J., Sapra, N. V., Petykiewicz, J. A., and Vučković, J. (2020). Nanophotonic inverse design with SPINS: Software architecture and practical considerations. *Applied Physics Reviews*, 7(1).

Svanberg, K. (2002). A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, pages 555–573.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Yang, J. and Fan, J. A. (2017). Topology-optimized metasurfaces: impact of initial geometric layout. *Opt. Lett.*, 42(16):3161–3164.