EDA 软件设计 I

Lecture 19

数字电路基础

Over

Course Outline

- **Chapter 1: Introduction to EDA**
- **Chapter 2: EDA常用图算法** 重点
- **Chapter 3: 数字电路基础**
- **Chapter 4: 优化算法、数学规划**

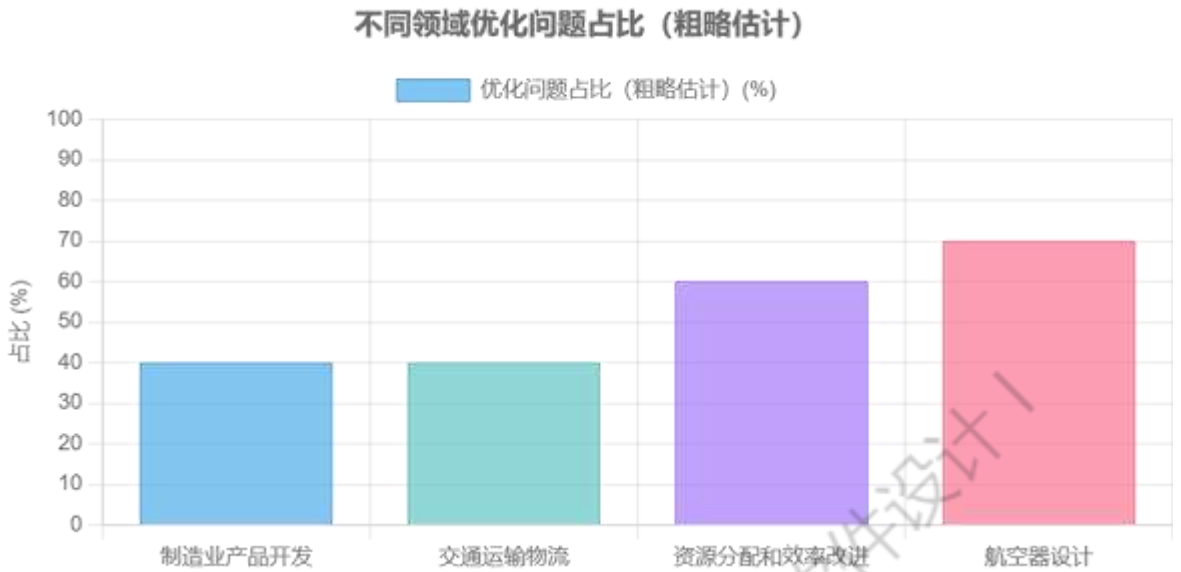
重点、难点

优化问题：工程问题中占比很大的课题

60% ~ 70%

无处不在的优化问题

1. **复杂项目的标准流程：**
   - 在工程设计或开发流程中，通常需要多个阶段的优化
2. **资源分配和效率改进：**
   - 工程问题往往围绕着有限资源的分配展开，优化方法在资源分配和效率改进中起到至关重要的作用
3. **高端技术中的核心作用：**
   - 在尖端领域（如航空航天、人工智能结合工程），优化方法往往是整个系统设计和分析的核心



优化问题出现多的原因

- **工程问题的复杂性**：工程设计通常有多个目标和约束条件（多目标优化问题）
- **资源有限性**：资金、材料、时间、能源等有限资源的分配需要优化
- **效率需求**：在工程系统中，效率通常是决定成败的关键因素，优化能够极大提高效率
- **竞争性压力**：工程问题中的优化往往决定了设计的经济性和市场竞争力

科研论文中的优化算法

Algorithm 2: Entanglement Path Selection (EPS)

Input: CPES $\{r_{ijl}\}$, algorithm approximation ratio requirement ϵ

Output: Entanglement path selection to maximize efficient network throughput $\{x_{ij}\}$

Based on $\{r_{ijl}\}$, formulate Problem (12)

Relax constraint (12c) to be $x_{ij} \geq 0$ and solve derived LP model, say the solution is $\hat{x}_{ij}$ and corresponding objective value is $\hat{z}$

Let $s \leftarrow \min\{\lfloor \hat{z} \rfloor, \frac{m(1-\epsilon)}{\epsilon}\}$, where m is the number of constraints in (12a) and (12b)

Initialize $z^{ALG} \leftarrow 0$, $x_{ij}^{ALG} \leftarrow 0$

for t *from* $\sum_{i,j} \lfloor \hat{x}_{ij} \rfloor$ *to* s **do**

  for *each solution* $T = \{\bar{x}_{ij}\}$ *satisfying* $\sum_{i,j} \bar{x}_{ij} = t$ **do**

    Substitute constraint (12c) to be $x_{ij} \geq \bar{x}_{ij}$ and solve the derived LP, say the solution is x_{ij}^*

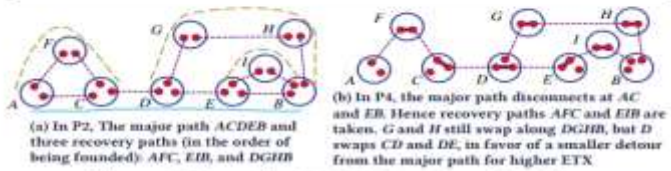
    if *the derived LP is feasible and* $z^{ALG} < \sum_{i,j} \lfloor x_{ij}^* \rfloor$ **then**

      $z^{ALG} \leftarrow \sum_{i,j} \lfloor x_{ij}^* \rfloor$, $x_{ij}^{ALG} \leftarrow x_{ij}^*$

Return $\{x_{ij}^{ALG}\}$ and z^{ALG}

Algorithm 2: The Extended Dijkstra's algorithm

Input: $G = \langle V, E, C \rangle$, e, $\langle src, dst \rangle$
Output: The best path $\langle p, W \rangle$
// Initialize empty states
$E \leftarrow$ an array of n elements, all set to $-\infty$
$prev \leftarrow$ an array of n elements, all set to null
$visited \leftarrow$ an array of n elements, all set to false
$width \leftarrow$ an array of n elements, all set to 0
$q \leftarrow$ fibonacci-heap, highest $E[\cdot]$ first
// Initialize states of src
$E[src] \leftarrow +\infty$
$width[src] \leftarrow +\infty$
q.enqueue(src)
// Track the best path until dst
while q is not empty **do**
  // Get the current best end node
  $u \leftarrow q$.dequeue()
  if $visited[u]$ **then continue**
  else $visited[u] \leftarrow$ true
  if $u = dst$ **then**
    $\langle p, W \rangle \leftarrow$ Construct path via $prev$ and $width$
    return $\langle p, W \rangle$
  // Expand one hop based on u
  for $v \in$ neighbors of u **do**
    if $visited[v]$ **then continue**
    $\langle p, W \rangle \leftarrow$ Construct path via $prev$ and $width$
    $E' \leftarrow e(p, W)$
    if $E[v] < E'$ **then**
      $E[v] \leftarrow E$
      $prev[v] \leftarrow u$
      $width[v] \leftarrow W$
      q.reorder(v)



(a) In P2, The major path *ACDEB* and three recovery paths (in the order of being founded): *AFC*, *EIB*, and *DGHB*

(b) In P4, the major path disconnects at *AC* and *EB*. Hence recovery paths *AFC* and *EIB* are taken. *G* and *H* still swap along *DGHB*, but *D* swaps *CD* and *DE*, in favor of a smaller detour from the major path for higher ETX

Figure 9: Example of Q-CAST recovery algorithm

Algorithm 3: Online Top-K Path Selection Algorithm

Input: $\mathcal{L}$, $\mathcal{M}_{\text{init}}$, N_{rep}, $\mathcal{M}_{\text{loop}}$, L, K, δ, h_1, h_2
Output: high-fidelity paths and fidelity information
$b_{i,m} \leftarrow$ Average (*Algorithm 1*(i, m), N_{rep}), $\forall i \in \mathcal{L}, m \in \mathcal{M}_{\text{init}}$
$A_i, \hat{p}_i \leftarrow$ Regression $(\mathcal{M}_{init}, \{b_{i,m}\}_{m \in \mathcal{M}_{init}})$ for $i \in \mathcal{L}$
$\hat{p}_i(1) \leftarrow \hat{p}_i$ for $i \in \mathcal{L}$; $\mathcal{L}(1) \leftarrow \mathcal{L}$; $\mathcal{L}_{\text{good}} \leftarrow \varnothing$; $\mathcal{L}_{\text{bad}} \leftarrow \varnothing$
for $t = 1, 2, \cdots$ **do**
  $\text{UCB}_i(t) \leftarrow \hat{p}_i(t) + \sqrt{\log(4Lt^2/\delta)/(2t)}$ for $i \in \mathcal{L}(t)$
  $\text{LCB}_i(t) \leftarrow \hat{p}_i(t) - \sqrt{\log(4Lt^2/\delta)/(2t)}$ for $i \in \mathcal{L}(t)$
  $\mathcal{H}(t) \leftarrow K - |\mathcal{L}_{\text{good}}|$ paths in $\mathcal{L}(t)$ with the highest UCBs
  for $i \in \mathcal{L}(t) \setminus \mathcal{H}(t)$ **do**
    if $\min_{j \in \mathcal{H}(t)} \text{LCB}_j(t) > \text{UCB}_i(t)$ **then**
      $\mathcal{L}_{\text{bad}} \leftarrow \mathcal{L}_{\text{bad}} \cup \{i\}$
  for $i \in \mathcal{H}(t)$ **do**
    if $\text{LCB}_i(t) > \max_{j \in \mathcal{L}(t) \setminus \mathcal{H}(t)} \text{UCB}_j(t)$ **then**
      $\mathcal{L}_{\text{good}} \leftarrow \mathcal{L}_{\text{good}} \cup \{i\}$
  $\mathcal{L}(t) \leftarrow \mathcal{L}(t) \setminus (\mathcal{L}_{\text{good}} \cup \mathcal{L}_{\text{bad}})$
  if $|\mathcal{L}_{\text{good}}| \geq K$ or $|\mathcal{L}_{\text{bad}}| \geq L - K$ **then return** $\mathcal{L}_{\text{good}}$
  if $\min_{i \in \mathcal{H}(t)} \text{LCB}_i(t) > h_1$ or $\max_{i \in \mathcal{L}(t) \setminus \mathcal{H}(t)} \text{UCB}_i(t) < h_2$ **then return** $\mathcal{L}_{\text{good}} \cup \mathcal{H}(t)$
  for $i \in \mathcal{L}(t)$ **do**
    $m_i \leftarrow \arg\max_{m \in \mathcal{M}_{\text{loop}}} I(\hat{p}_i(t), m)$
    $b_{i,m_i}(t) \leftarrow$ Algorithm $1(i, m_i)$
    $p_i(t) \leftarrow (b_{i,m_i}(t)/A_i)^{1/(2m_i)}$
    $\hat{p}_i(t+1) \leftarrow (\hat{p}_i(t) * t + p_i(t))/(1+t)$
    Update A_i via regression
  $\mathcal{L}(t+1) \leftarrow \mathcal{L}(t)$

a large quantum resource consumption. We refer interested readers to Appendix A for the proof.

Leetcode题目大多也涉及优化

- 300. 最长递增子序列
- 1143. 最长公共子序列
- 435. 无重叠区间
- 452. 用最少数量的箭引爆气球
- 455. 分发饼干
- 787. K 站中转内最便宜的航班

EDA参与芯片设计相关优化

- **EDA领域，优化是一个多层次、多维度的过程，涉及逻辑、物理、功耗、时序等多个方面**
- **逻辑优化**
  - 布尔代数简化:使用布尔代数和卡诺图等方法简化逻辑表达式，以减少逻辑门的数量和复杂性
  - 逻辑综合：将高层次的设计描述（如VHDL或Verilog）转换为门级实现，优化过程中会考虑到逻辑门的数量和延迟
- **物理设计优化**
  - 布局优化：优化电路的布局以减少信号延迟和功耗，通常涉及电路元件的放置和连线
  - 布线优化：优化布线以减少布线长度、降低电阻和电容，从而提高信号完整性和降低功耗
- **时序优化：**确保信号在规定的时序约束内到达，通常涉及到对路径延迟的优化
  - 时序分析、时序修复、多域时钟设计
- **自适应优化：**通过机器学习技术实现自适应优化，根据历史数据动态调整优化策略

Chapter 4: 优化 Optimization

- 贪心 (Greedy)
- 动态规划 (Dynamic Programming)
- 启发式算法 (Heuristic)
- 数学规划 (Mathematical Programming)

第四章内容：优化算法与数学规划

属性	贪心、动态规划、启发式（算法原理）	数学规划（方法）
定位	一种算法设计思想	一种数学建模和求解方法
应用场景	广泛（不限于数学优化问题）	通常用于数学优化问题
结果质量	根据具体问题而定	找到最优解（理论保证）
依赖数学建模	不一定	是数学规划的核心

In this class

- Intro to Optimization (优化)
  - 什么是优化问题（通常具有的结构）
  - 优化问题的类型
  - 优化问题的（常见）解法
- 贪心算法

什么是优化问题

What is an optimization problem

优化（Optimization）问题

优化问题是：数学和工程领域中常见的问题，指的是在一定约束条件下，寻找使目标函数达到最大值或最小值的解

◆优化问题的核心在于"优化"，即找到最优解（最大化或最小化某一目标）

- **优化问题给定 （input）：**
  - ① 一个**问题实例（problem instance)**
  - ② 一组**约束条件（constraints）：指定了所需解决方案的限制**
  - ③ 一个**目标函数 (objective function）：需要最大化或最小化的函数**
- **优化问题目标 （output）：**为给定问题实例找到可行的解决方案（feasible solution), 要么是最大值要么是最小值，取决于要解决的问题

优化问题重要组成要素：目标函数、约束条件

属性	数学规划	贪心/动态规划
目标函数	明确写出，如 $\max z = c_1x_1 + c_2x_2$	隐含在选择规则（贪心）或状态转移中（动态规划）
约束条件	形式化表示，如 $a_1x_1 + a_2x_2 \leq b$	在算法逻辑中体现，不一定单独表达
表达方式	数学符号化，适合用求解器处理	算法化，直接设计求解过程

优化问题类型

给优化问题分类的原因 (benefit)：

- 帮助问题从“一个大而复杂的难题”变成一个“特定的小的问题”
- “找到最适合的方法来高效求解”——使用现有的优化工具

不同的分类维度

① 按目标函数分类

- 线性优化和非线性优化问题
- 单目标优化问题和多目标优化问题

② 按（决策）变量分类

- 连续优化问题：变量可以取值连续（如实数）
- 离散优化问题：变量只能取整数或特定离散值

③ 按约束条件分类

- 有约束优化问题（等式或不等式）
- 无约束优化问题

④ 按解法分类

- 数学规划：解析法（确定性、理论保证最优解）、数值法（最优解的近似解）
- 优化算法：贪心、动态规划、启发式

最常见的优化问题类型

① **线性规划（Linear Programming，LP）**

- 目标函数和约束条件都是线性函数

② **非线性优化（Non-linear Optimization, NLP）**

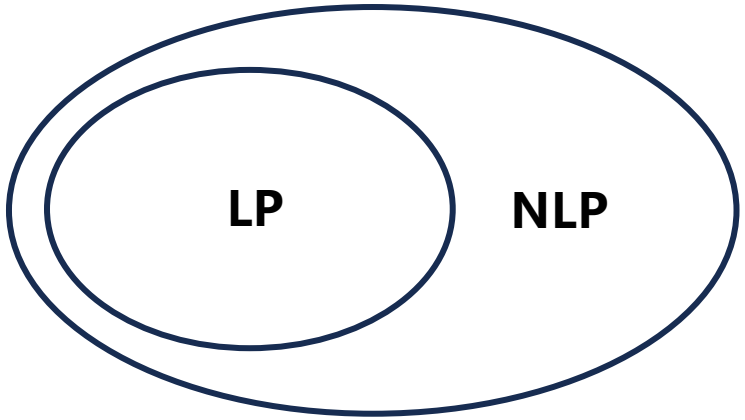
- 目标函数或约束条件包含非线性项
- 例子: Minimize $x_1^2 + x_2^2$，subject to $x_1 + x_2 = 1$

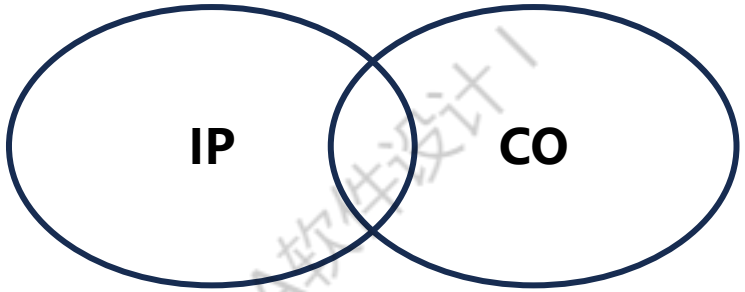
③ **整数规划（Integer Programming, IP）**

- 决策变量必须是整数（或部分取整数）

④ **组合优化 （Combinatorial Optimization, CO）**

- 在有限的离散结构中寻找最优解的问题，通常涉及选择、排列或组合的决策





优化问题分类的原因

Benefit 1: 利用问题的特殊结构来设计优化算法，例如:

- 凸优化（Convex Optimization）：如果目标函数和约束是凸函数，就可以保证找到全局最优解，且可以使用高效的算法
- 非凸优化（Non-Convex Optimization）：非凸问题可能有多个局部最优点，需要特殊的方法（如随机优化）

Benefit 2：更系统地分析问题的性质和复杂性，例如:

- 线性规划属于P类问题（可多项式时间内求解）
- 整数规划属于NP-hard问题（一般无法保证多项式时间内求解）
- 组合优化问题中既有P类问题，也有NP-hard问题

优化问题分类的原因

Benefit 3: 辅助问题建模，例如：

- 如果问题中涉及连续变量，可以建模为线性或非线性规划
- 如果涉及选择或布尔变量，则更适合整数规划或组合优化

Benefit 4: 方便匹配现有优化工具：

- 线性规划规划工具：如**Gurobi**、CPLEX（IBM）、**Python SciPy**
- 非线性规划工具：如IPOPT、**MATLAB Optimization Toolbox**
- 凸优化工具：**cvx (开源Matlab插件)**
- 组合优化和整数规划工具：如OR-Tools （Google）、COIN-OR

优化问题的解法

优化问题的解决方法

数学规划

- **解析法**
  - 定义：可以通过**数学推导**得到**精确解**（**理论保障最优**）
  - 例如：拉格朗日乘数法解决有约束优化问题
- **数值法**
  - 定义：通过**近似计算**的方法，使用离散的数值步骤解决数学问题
  - 例如：梯度下降法（Gradient Descent）、牛顿法（Newton's Method）

优化算法

- **确定性算法**
  1. **动态规划**：通过分解问题为子问题并存储子问题的解来优化求解过程的算法设计方法
  2. **贪心算法**：通过在每一步选择当前最优选项来构建最终解的方法
- **启发式算法**
  - 一种利用经验规则或问题特性，快速找到问题近似解的算法
  - 例如：遗传算法、模拟退火算法

确定性算法：贪心和动态规划

特性	动态规划（DP）	贪心算法（Greedy）
核心思想	分解为子问题，存储和重用中间结果	每一步选择当前最优解
适用问题结构	子问题重叠性、最优子结构	贪心选择性质、最优子结构
解法保证	**始终保证全局最优解**	**不一定保证全局最优解**
时间复杂度	通常较高，取决于子问题数量	通常较低，每步计算一次
空间复杂度	较高（需要存储子问题的解）	较低（通常只存储当前状态）
实现复杂性	实现复杂，涉及递归或迭代及表格管理	实现简单，直接选择