OS01 project:
Lagrangian relaxation for the travelling salesman problem

1. General principle of Lagrangian relaxation.

Consider a mathematical program (P) with a vector x of n variables and m constraints. f is the objective function and $g_i(x)$ are the constraint functions. S is a subset of $\mathbb{R}^n$ defined by other constraints. For example, for an integer linear programming, f and g_i are linear and $S = \mathbb{N}^n$.

$$(P) \begin{cases} min\, z = f(x) \\ g_i(x) \leq 0,\ i = 1, \dots m \\ x \in S \end{cases}$$

Lagrangian relaxation (LR) consists of multiplying the $g_i(x)$ by the weights π_i and adding them to z. The weights are called *Lagrange multipliers* or *dual variables*. They are real positive or zero for constraints of inequality, and of any sign for constraints of equality. By denoting $g(x)$ the vector of the $g_i(x)$, we obtain the *Lagrangian function* $L(x, \pi) = f(x) + \pi.g(x)$. The *dual function* $L(\pi)$ gives the minimum of the Lagrangian function for π fixed, that is $L(\pi) = min_{x \in S} L(x, \pi)$. We call the computation of the dual function for fixed π a *Lagrangian problem*.

2. Solving the dual by a subgradient method.

We use a method of successive increases based on *subgradients*. Specifically, we choose a subgradient, we make a small displacement λ in its direction, then we use a new subgradient at the resulting point. The general structure of a subgradient algorithm is:

Algorithm 1

1. Initialize the multiplier vector π and the lower bound LB to zero.
2. Initialize the iteration number j to zero.
3. **Repeat**
4. $j \leftarrow j + 1$
5. Calculate the minimum of the dual function in π:
6. $L(\pi) = f(y) + \pi.g(y) = min_{x \in S} f(x) + \pi.g(x)$, ($\gamma(\pi) = g(y)$ is a subgradient of L in π).
7. **If** $L(\pi) > LB$, **then** $LB \leftarrow L(\pi)$ **end if.**
8. **If** $\|\gamma(\pi)\| > \epsilon$, **then**:
9. Determine the step λ_j for this iteration.
10. Calculate the next point: $\pi \leftarrow \pi + \lambda_j.\gamma(\pi) / \|\gamma(\pi)\|$.
11. For any inequality constraint i, force π_i to zero if it is negative.
12. Update the step λ.
13. **End if**.
14. **Until** ($j = j_{max}$) ou ($\|\gamma(\pi)\| < \epsilon$) ou ($LB = UB$) ou ($y$ is primal-achievable).

The dual function may temporarily decrease or oscillate, therefore it is better to update the lower bound (LB) every time it is improved (line 7).

There are four stopping cases (line 14). The first is a maximum number of iterations j_{max} when the convergence is too slow. The second is when the norm of the subgradient becomes less than a precision ϵ: the solution of the dual is then optimal, but not necessarily for the primal because of the duality jump. The third case is the attainment of an upper bound (UB) provided by a heuristic: we know then that the bound and the heuristic are optimal. Finally, for some problems, the solution of the Lagrangian problem can meet all the constraints of (P). It is therefore optimal for (P) and we can stop the subgradient method.

Here are three examples of classic rules to update the step λ_j used at iteration j:

R1 - Constant step:
This is the simplest rule: the λ_j are all equal to a positive constant λ, for example $\lambda = 1$.

R2 - Rule of divergent series:
It is showed that $L(\pi)$ converges towards the optimum if λ_j form a divergent sequence:
$\lambda_j \to 0 \, (j \to +\infty)$ and $\sum_{j=1}^{\infty} \lambda_j = +\infty$
An simple example is $\lambda_j = 1/j$. The convergence is sometimes very slow.

R3 — Rule of convergent series:
This rule works well in practice but can converge to a non-optimal point. We utilize $\lambda_j = \sigma.\alpha^j$, where σ is a positive constant and $0 < \alpha < 1$. To avoid a convergence towards a non-optimal point, it is better off taking α fairly close to 1 and σ rather large.

3. Lagrangian relaxation for the traveling salesman problem.

This is the first application of LR to combinatorial optimization. Previously, the method was only used in nonlinear optimization. The traveling salesman problem (TSP) is defined by considering a complete non-oriented graph G of n nodes, with a matrix C of size $n \times n$ giving the costs of edges. We need to find a minimum cost cycle that starts from the depot (node 1), visits each node once, and returns to the depot.

A cycle-free graph that connects all nodes is called a *spanning tree*. The computation of a spanning tree G is possible with Prim's algorithm. At each iteration, A is the set of edges in the tree, S the set of connected nodes, and h the total cost (distance):

Algorithm 2

1. Start from a tree reduced to node 1: $S \leftarrow \{1\}$, $A \leftarrow \emptyset$, $h \leftarrow 0$.
2. **For** iteration 1 **to** n - 1 **do**:
3. Determine the edge $[i, j]$ of minimal cost, with $i \in S$ and $j \notin S$
4. $h \leftarrow h + C_{ij}$
5. $S \leftarrow S \cup \{j\}$
6. $A \leftarrow A \cup \{[i, j]\}$
7. **End for**.

The degree $D(i)$ of a node i in a graph is the number of edges incident to i. A *1-tree* is a tree spanning nodes from 2 to n, plus two edges incident to node 1. It is easy to see that a 1-tree is formed by a cycle passing through node 1, with possibly some trees grafted to this cycle. A solution of the TSP is a special case of 1-tree, where all nodes are on the cycle and thus all have degree 2.

Therefore, the cost of an optimal 1-tree is a lower bound of the optimal cost for the TSP. Moreover, if the optimal 1-tree has all its nodes of degree 2, it is also the optimal solution for the TSP.

To compute an optimal 1-tree, it is sufficient to adapt Prim's algorithm:

Algorithm 3

1. Compute a spanning tree on nodes 2 to n with Prim's algorithm (starting at node 2).
2. Add the two smallest incident edges to node 1.