Simple policy iteration

Monday, 5 September 2022 10.33

The algorithm takes as input

$A \in \mathbb{R}^{n \times n}$

$b \in \mathbb{R}^n$

$\lambda \in \mathbb{R}$ such that $\lambda > 0$

$R \subseteq [n] \times [n]$ a BDB

The BDB is used to make sure that solutions that are already known to be equivalent will be treated as such from the very start of the algorithm. Just to speed-up the algorithm

In many case studies we simply pass as input the largest BDB

The output of the algorithm will be a matrix $D \in \mathbb{R}^{n \times n}$ representing an invariant as explained in the draft

The whole algorithm works around the concept of policy

In the paper we present it as a map $\pi : [n] \times [n] \to \mathbb{R}^{n \times n}$ such that for all $i,j \in [n]$ $\pi(i,j) \in \Gamma(A_i, A_j)$

You can see a policy as a real-valued array that has 4 dimensions, each of them idexed in $\{1, \dots, n\}$.

Each policy is associated to a matrix $\delta_\lambda^\pi \in \mathbb{R}^{n \times n}$ which represents an invariant.

We compute δ_λ^π as the optimal solution of the following linear program.

Compute δ_λ^π

$$\arg\min_{D} \sum_{i=1}^{n} \sum_{j=1}^{n} D_{ij}$$

subject to the following constraints

$$\forall i,j = 1..n \quad D_{ij} \geq 0$$

$$\forall i,j = 1..n \quad \sum_{\substack{h,k \in [n] \\ st(h,k) \neq (i,j)}} \pi(i,j)_{h,k} D_{hk} + (\pi(i,j)_{ij} - 1) D_{ij} \leq \left[2\lambda \sum_{h,k \in [n]} \pi(i,j)_{hk}\right] - |b_i - b_j| - \left[\lambda \sum_{k \in [n]} |A_{ik}| + |A_{jk}|\right]$$

- constants
- variables
- coefficients

For the sake of the implementation here we can use any solver. We do not need to have to use solutions that are vertices.

The algorithm iteratively update the current policy ensuring that its corresponding invariant has decreased with respect to the previous one.

The update is performed by solving a simple variant of a transportation problem.

Given a matrix of costs $D \in \mathbb{R}^{n \times n}$ and $c, d \in \mathbb{R}^n$ the Transport schedule map $K_\lambda(D)(c,d)$ computes the optimal solution $W \in \mathbb{R}^{n \times n}$ resulting from linear program

Compute $K(D)(c,d)$

$$\arg\min_{s, \bar{s}, W} \sum_{i=1}^{n} s_i + \sum_{j=1}^{n} \bar{s}_j + \sum_{i=1}^{n} \sum_{j=1}^{n} D_{ij} W_{ij}$$

subject to the following constraints

$$\sum_{j=1}^{n} W_{ij} + s_i = c_i^+ + d_i^- \qquad \forall i = 1..n$$

$$\sum_{i=1}^{n} W_{ij} + \bar{s}_j = c_j^- + d_j^+ \qquad \forall j = 1..n$$

$$W_{ij} \geq 0 \qquad \forall i,j = 1..n$$

$$s_i \geq 0 \qquad \forall i = 1..n$$

$$\bar{s}_j \geq 0 \qquad \forall j = 1..n$$

Note that the optimal value of this linear program corresponds to

$T_\lambda(D)(c,d)$

as defined in the draft.

In contrast to the previous linear program, the solutions that have to be used here must be vertices.

This is ensured for example if the solver implements the simplex method.

Finally we have all the ingredients describe the simple policy iteration algorithm

```
SimplePolicyIteration (A, b, λ, R)
  // Construct an initial policy π
  for i = 1 to n
    for j = 1 to n
      if (i,j) ∈ R
        π[i,j] = K_λ(D_R)(A_i, A_j)
      else
        π[i,j] = 0
  // Iterative policy improvement
  Compute δ_λ^π
  While ∃(i,j). Δ_λ(δ_λ^π)_ij < (δ_λ^π)_ij
    // we update the policy at (i,j), the rest remains as it was
    π[i,j] = K_λ(δ_λ^π)(A_i, A_j)
    // prepare for the next iteration
    Compute δ_λ^π
  // after exiting the loop δ_λ^π is a fixed point for Δ_λ
  Return δ_λ^π
```

here $(D_R)_{ij} = \begin{cases} 0 & \text{if } (i,j) \in R \\ 1 & \text{if } (i,j) \notin R \end{cases}$

The matrix with zeros

Recall that $(\Delta_\lambda(D))_{ij} = |b_i - b_j| + T_\lambda(D)(A_i, A_j)$

Therefore the condition in the while loop can be rewritten as

$$|b_i - b_j| + T_\lambda(\delta_\lambda^\pi)(A_i, A_j) < (\delta_\lambda^\pi)_{ij}$$

Since the value of $T_\lambda(\delta_\lambda^\pi)(A_i, A_j)$ corresponds to the optimal value computed by solving the linear program for $K_\lambda(\delta_\lambda^\pi)(A_i, A_j)$, for the update you can simply reuse the optimal solution that you find while checking the condition of the while loop.