Solving most cost-effective loan problem

Rodion “rodde” Efremov

March 6, 2018

1 Introduction

In the most cost-effective loan problem, we are given a directed graph of actors where each actor may lend some amount of resources it possesses to its child nodes. In case an actor needs more than his immediate parents can lend, the parents might need to lend from their parents, adjust the interest rate to cover their own expenses, and pass the funds to the original lending actor.

Formally, we are given a directed graph $G = (V, A)$, where V is the set of actors, and $A \subseteq V^2$ is the set of directed arcs. By $V(G)$ we denote the actor set of G, and likewise, by $A(G)$ we denote the arc set of G. Given an arc $(u, v) \in A$, we call u a *parent* of v and v a *child* of u. Existence of such an arc indicates that u may lend some or all of its resources to v. Along the graph, we are given a *potential function* $\mathfrak{P}\colon V \to [0, \infty) = \mathbb{R}_{\geq 0}$ that maps each actor in the graph to the (non-negative) equity that that very node has at its disposal. Finally, we are given an *interest rate function* $\mathfrak{I}\colon A \to \mathbb{R}_{\geq 0}$ that maps each arc $(u, v) \in A$ to the interest rate the actor u can offer v when v decides to lend from u.

Apart from the target data structure, in a problem instance, we are given an actor $a \in V$ that applies for a loan, a required potential $P \in \mathbb{R}_{\geq 0}$, and a maximum tolerable interest rate $i \in \mathbb{R}_{\geq 0}$. Our aim, then, is to compute a loan (which may involve more than one lending actor) with minimized interest rate. Note that if actors would simply “pass” the potential from their parents to the lending actor, the problem would become trivial.

2 Interest rate model

Throughout the paper, we assume a simple interest rate model. The accumulated balance at time t since the time point at which the loan was issued, with initial principal $\mathfrak{A}$ and interest rate r is given by

$$\mathfrak{A}(1+r)^t.$$

If we have in the graph, for instance, a directed (acyclic) path (u, v, z) with $r_{u,v}$ being the interest rate of (u, v), and $r_{v,z}$ being the interest rate of (v, z), the interest rate equation becomes

$$\mathfrak{A}(1+r_{u,v})^t(1+r_{v,z})^t = \mathfrak{A}\big[(1+r_{u,v})(1+r_{v,z})\big]^t = \mathfrak{A}(1+R)^t.$$

Above, R is the *combined* interest rate. Dividing the both sides by $\mathfrak{A}$ and taking the t-th root, we obtain

$$\begin{aligned}(1+r_{u,v})(1+r_{v,z}) &= 1+R \\ R+1 &= 1+r_{u,v}+r_{v,z}+r_{u,v}r_{v,z} \\ R &= r_{u,v}+r_{v,z}+r_{u,v}r_{v,z}.\end{aligned}$$

In general, we write $r_1 + r_2 + r_1 r_2 = \mathfrak{C}(r_1, r_2)$.

Since we may deal with entire “loan chains”, we need to define the concept of *effective* interest rate. Effective interest rate is given by

$$I(u,v) = \begin{cases} 0 & \text{if } u = v, \\ \min\limits_{z \in \textsc{Children}(G,u)} \mathfrak{C}\big(\mathfrak{I}(u,z), I(z,v)\big) & \text{otherwise,} \end{cases}$$

where $\textsc{Children}(G, z)$ is the set of child nodes of z, or, formally, $\{u\colon (z,u) \in A(G)\}$.

3 Problem statement

Given a problem instance $(G, a, \mathfrak{P}, \mathfrak{I}, P, i)$, we wish to compute two additional functions π and d. $\pi\colon V \to \mathbb{R}_{\geq 0}$ is called a *solution potential function* and it maps each actor u to potential $\pi(u)$ u can lend, and $d\colon V \to V$ is called a *direction function* and it maps each actor u to some of its children $d(u)$ to which $\pi(u)$ worth potential is being lent. What comes to constraints, no actor u lending some of its potential shall have $I(u, a) > i$, since a cannot afford effective interest rates above i. Also, if it is not possible due to the first constraint to obtain a loan worth P, the loan should be maximized from below as close to P as possible.

In order to implement the first constraint, we need to define the set of *admissible* solution potential functions:

$$\pi_{I,a,i,G} = \{\pi \colon V(G) \to \mathbb{R}_{\geq 0} \mid \pi(u) = 0 \text{ if } I(u, a) > i\}.$$

An admissible solution potential function π is said to be *valid* if it also satisfies

$$\sum_{u \in V} \pi(u) \in [0, P],$$

and we denote that fact by $\pi \in \mathfrak{V}$.

Now, we can state the objective formally:

$$\pi = \operatorname*{arg\,min}_{\pi' \in \mathfrak{V}} \left[P - \sum_{u \in V} \pi'(u) \right],$$

and

$$d(u) = \operatorname*{arg\,min}_{z \in \textsc{Children}(G,u)} \mathfrak{C}\big(\mathfrak{I}(u, z), I(z, a)\big).$$

4 Solution algorithm

Since the effective interest rate does not decrease with adding more directed arcs, we must have that for any actor $a \in V$ the most cost-efficient lenders are its immediate parents. This observation implies that we basically wish to compute a directed "minimum spanning tree" in which each other node in the tree has a single unique directed path to a. In the very beginning, the tree is trivial and consists of only a. Then a most cost-effective parent u_1 is selected and the arc (u_1, a) is introduced. Then u_2 is selected. It must not belong to $\{a, u_1\}$ while have the lowest possible effective interest rate among all nodes in $V \setminus \{a, u_1\}$. This procedure continues until a desired potential P is collected or until there is no more nodes left with affordable effective interest rates. Below, $\textsc{Priority-Queue-Insert}(Q, \langle a, b, c \rangle)$ stores the triple $\langle a, b, c \rangle$ and uses c as the priority key.

Algorithm 1: Most-Cost-Effective-Loans($G, a, \mathfrak{P}, \mathfrak{I}, P, i$)

```
let Q be an empty priority queue
let π be an empty solution potential function
let d be an empty direction function
C ← ∅
P_current ← 0
for u ∈ V(G) do
    π(u) ← 0
    d(u) ← nil
for u ∈ Parents(G, a) do
    if 𝔍(u, a) ≤ i then
        Priority-Queue-Insert(Q, ⟨u, a, 𝔍(u, a)⟩)
while |Q| > 0 and P_current < P do
    ⟨u, v, i_current⟩ ← Priority-Queue-Extract-Minimum(Q)
    P_Δ ← min(P − P_current, 𝔓(u))
    P_current ← P_current + P_Δ
    π(u) ← P_Δ
    d(u) ← v
    C ← C ∪ {u}
    for z ∈ Parents(G, u) do
        if z ∉ C then
            i_next ← ℭ(i_current, 𝔍(z, u))
            if i_next ≤ i then
                Priority-Queue-Insert(Q, ⟨z, u, i_next⟩)
    return (π, d)
```