Optimal partition assignment in Garage

Mendes Oulamara

July 18, 2022

1 Introduction

1.1 Context

Garage is an open-source distributed storage service blablabla...

Every object to be stored in the system falls in a partition given by the last k bits of its hash. There are $N = 2^k$ partitions. Every partition will be stored on distinct nodes of the system. The goal of the assignment of partitions to nodes is to ensure (nodes and zone) redundancy and to be as efficient as possible.

1.2 Formal description of the problem

We are given a set of nodes V and a set of zones Z. Every node v has a non-negative storage capacity $c_v \geq 0$ and belongs to a zone $z_v \in Z$. We are also given a number of partition $N > 0$ (typically $N = 256$).

We would like to compute an assignment of three nodes to every partition. That is, for every $1 \leq i \leq N$, we compute a triplet of three distinct nodes $T_i = (T_i^1, T_i^2, T_i^3) \in V^3$. We will impose some redundancy constraints to this assignment, and under these constraints, we want our system to have the largest storage capacity possible. To link storage capacity to partition assignment, we make the following assumption:

$$\textit{All partitions have the same size } s. \tag{H1}$$

This assumption is justified by the dispersion of the hashing function, when the number of partitions is small relative to the number of stored large objects.

Every node v needs to store $n_v = \#\{1 \leq i \leq N \mid v \in T_i\}$ partitions (where # denots the number of indices in the set). Hence the partitions stored by v (and hence all partitions by our assumption) have there size bounded by c_v/n_v. This remark leads us to define the optimal size that we will want to maximize:

$$s^* = \min_{v \in V} \frac{c_v}{n_v}. \tag{OPT}$$

When the capacities of the nodes are updated (this includes adding or removing a node), we want to update the assignment as well. However, transferring the data between nodes has a cost and we would like to limit the number of changes in the assignment. We make the following assumption:

Updates of capacity happens rarely relatively to object storing. (H2)

This assumption justifies that when we compute the new assignment, it is worth to optimize the partition size (OPT) first, and then, among the possible optimal solution, to try to minimize the number of partition transfers.

For now, in the following, we ask the following redundancy constraint:

Mode 3-strict: every partition needs to be assignated to three nodes belonging to three different zones.

2 Properties of an optimal 3-strict assignment

2.1 Optimal assignment

For every zone $z \in Z$, define the zone capacity $c_z = \sum_{v, z_v = z} c_v$ and define $C = \sum_v c_v = \sum_z c_z$.

One can check that the best we could be doing to maximize s^* would be to use the nodes proportionally to their capacity. This would yield $s^* = C/(3N)$. This is not possible because of (i) redundancy constraints and (ii) integer rounding but it gives and upper bound.

Optimal utilization

We call an *utilization* a collection of non-negative integers $(n_v)_{v \in V}$ such that $\sum_v n_v = 3N$ and for every zone z, $\sum_{v \in z} n_v \leq N$. We call such utilization *optimal* if it maximizes s^*.

We start by computing a node sub-utilization $(\hat{n}_v)_{v \in V}$ such that for every zone z, $\sum_{v \in z} \hat{n}_v \leq N$ and we show that there is an optimal utilization respecting the constraints and such that $\hat{n}_v \leq n_v$ for every node.

Assume that there is a zone z_0 such that $c_{z_0}/C \geq 1/3$. Then for any $v \in z_0$, we define

$$\hat{n}_v = \left\lfloor \frac{c_v}{c_{z_0}} N \right\rfloor .$$

This choice ensures for any such v that

$$\frac{c_v}{\hat{n}_v} \geq \frac{c_{z_0}}{N} \geq \frac{C}{3N}$$

which is the universal upper bound on s^*. Hence any optimal utilization (n_v) can be modified to another optimal utilization such that $n_v \geq \hat{n}_v$

Because z_0 cannot store more than N partition occurences, in any assignment, at least $2N$ partitions must be assignated to the zones $Z \setminus \{z_0\}$. Let $C_0 = C - c_{z_0}$. Suppose that there exists a zone $z_1 \neq z_0$ such that $c_{z_1}/C_0 \geq 1/2$. Then, with the same argument as for z_0, we can define

$$\hat{n}_v = \left\lfloor \frac{c_v}{c_{z_1}} N \right\rfloor$$

for every $v \in z_1$.

Now we can assign the remaining partitions. Let $(\hat{N}, \hat{C})$ to be

- $(3N, C)$ if we did not find any z_0;
- $(2N, C - c_{z_0})$ if there was a z_0 but no z_1;
- $(N, C - c_{z_0} - c_{z_1})$ if there was a z_0 and a z_1.

Then at least $\hat{N}$ partitions must be spread among the remaining zones. Hence s^* is upper bounded by $\hat{C}/\hat{N}$ and without loss of generality, we can define, for every node that is not in z_0 nor z_1,

$$\hat{n}_v = \left\lfloor \frac{c_v}{\hat{C}} \hat{N} \right\rfloor .$$

We constructed a sub-utilization $\hat{n}_v$. Now notice that $3N - \sum_v \hat{n}_v \leq \#V$ where $\#V$ denotes the number of nodes. We can iteratively pick a node v^* such that

- $\sum_{v \in z_{v^*}} \hat{n}_v < N$ where z_{v^*} is the zone of v^*;
- v^* maximizes the quantity $c_v/(\hat{n}_v + 1)$ among the vertices satisfying the first condition (i.e. not in a saturated zone).

We iterate these instructions until $\sum_v \hat{n}_v = 3N$, and at this stage we define $(n_v) = (\hat{n}_v)$. It is easy to prove by induction that at every step, there is an optimal utilization that is pointwise larger than $\hat{n}_v$, and in particular, that (n_v) is optimal.

Existence of an optimal assignment

As for now, the *optimal utilization* that we obtained is just a vector of numbers and it is not clear that it can be realized as the utilization of some concrete assignment. Here is a way to get a concrete assignment.

Define $3N$ tokens $t_1, \ldots, t_{3N} \in V$ as follows:

- Enumerate the zones z of Z in any order;
- enumerate the nodes v of z in any order;
- repeat n_v times the token v.

Then for $1 \leq i \leq N$, define the triplet T_i to be (t_i, t_{i+N}, t_{i+2N}). Since the same nodes of a zone appear contiguously, the three nodes of a triplet must belong to three distinct zones.

However simple, this solution to go from an utilization to an assignment has the drawback of not spreading the triplets: a node will tend to be associated to the same two other nodes for many partitions. Hence, during data transfer, it will tend to use only two link, instead of spreading the bandwith use over many other links to other nodes. To achieve this goal, we will reframe the search of an assignment as a flow problem. and in the flow algorithm, we will introduce randomness in the order of exploration. This will be sufficient to obtain a good dispersion of the triplets.

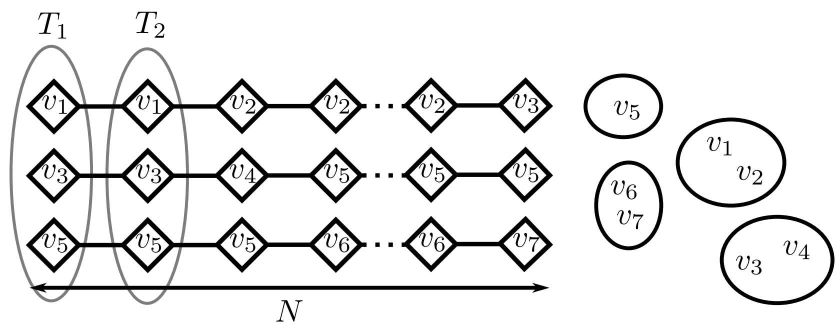

Figure 1: On the left, the creation of a concrete assignment with the naive approach of repeating tokens. On the right, the zones containing the nodes.

Assignment as a maximum flow problem

We describe the flow problem via its graph (X, E) where X is a set of vertices, and E are directed weighted edges between the vertices. For every zone z, define $n_z = \sum_{v \in z} n_v$.

The set of vertices X contains the source $\mathbf{s}$ and the sink $\mathbf{t}$; a vertex $\mathbf{x}_z$ for every zone $z \in Z$, and a vertex $\mathbf{y}_i$ for every partition index $1 \leq i \leq N$.

The set of edges E contains

- the edge $(\mathbf{s}, \mathbf{x}_z, n_z)$ for every zone $z \in Z$;
- the edge $(\mathbf{x}_z, \mathbf{y}_i, 1)$ for every zone $z \in Z$ and partition $1 \leq i \leq N$;
- the edge $(\mathbf{y}_i, \mathbf{t}, 3)$ for every partition $1 \leq i \leq N$.

We first show the equivalence between this problem and and the construction of an assignment. Given some optimal assignment (n_v), define the flow $f : E \to \mathbb{N}$ that saturates every edge from $\mathbf{s}$ or to $\mathbf{t}$, takes value 1 on the edge between $\mathbf{x}_z$ and $\mathbf{y}_i$ if partition i is stored in some node of the zone z, and 0 otherwise. One can easily check that f thus defined is indeed a flow and is maximum.

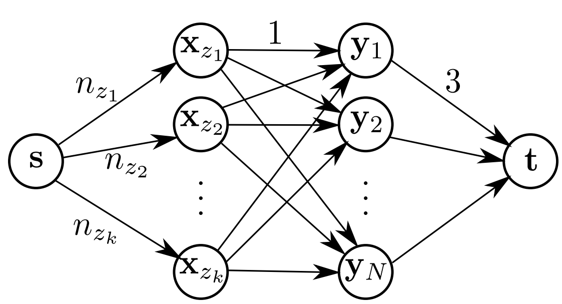

Figure 2: Flow problem to compute and optimal assignment.

Reciprocally, by the existence of maximum flows constructed from optimal assignments, any maximum flow must saturate the edges linked to the source or the sink. It can only take value 0 or 1 on the other edge, and every partition vertex is associated to exactly three distinct zone vertices. Every zone is associated to exactly n_z partitions.

A maximum flow can be constructed using, for instance, Dinic's algorithm. This algorithm works by discovering augmenting path to iteratively increase the flow. During the exploration of the graph to find augmenting path, we can shuffle the order of enumeration of the neighbours to spread the associations between zones and partitions.

Once we have such association, we can randomly distribute the n_z edges picked for every zone z to its nodes $v \in z$ such that every such v gets n_z edges. This defines an optimal assignment of partitions to nodes.

2.2 Minimal transfer

Assume that there was a previous assignment $(T'_i)_{1\leq i\leq N}$ corresponding to utilizations $(n'_v)_{v\in V}$. We would like the new computed assignment $(T_i)_{1\leq i\leq N}$ from some $(n_v)_{v\in V}$ to minimize the number of partitions that need to be transferred. We can imagine two different objectives corresponding to different hypotheses:

Transfers between different zones cost much more than inside a zone. (H3A)

Changing zone is not the largest cost when transferring a partition. (H3B)

In case A, our goal will be to minimize the number of changes of zone in the assignment of partitions to zone. More formally, we will maximize the quantity

$$Q_Z := \sum_{1\leq i\leq N} \#\{z \in Z \mid z \cap T_i \neq \emptyset, z \cap T'_i \neq \emptyset\}.$$

In case B, our goal will be to minimize the number of changes of nodes in the assignment of partitions to nodes. We will maximize the quantity

$$Q_V := \sum_{1\leq i\leq N} \#(T_i \cap T'_i).$$

It is tempting to hope that there is a way to maximize both quantity, that having the least discrepancy in terms of nodes will lead to the least discrepancy in terms of zones. But this is actually wrong! We propose the following counter-example to convince the reader:

We consider eight nodes $a, a', b, c, d, d', e, e'$ belonging to five different zones $\{a, a'\}, \{b\}, \{c\}, \{d, d'\}, \{e, e'\}$. We take three partitions ($N = 3$), that are originally assigned with some utilization $(n'_v)_{v\in V}$ as follows:

$$T'_1 = (a, b, c) \qquad T'_2 = (a', b, d) \qquad T'_3 = (b, c, e).$$

This assignment, with updated utilizations $(n_v)_{v\in V}$ minimizes the number of zone changes:

$$T_1 = (d, b, c) \qquad T_2 = (a, b, d) \qquad T_3 = (b, c, e').$$

This one, with the same utilization, minimizes the number of node changes:

$$T_1 = (a, b, c) \qquad T_2 = (e', b, d) \qquad T_3 = (b, c, d').$$

One can check that in this case, it is impossible to minimize both the number of zone and node changes.

Because of the redundancy constraint, we cannot use a greedy algorithm to just replace nodes in the triplets to try to get the new utilization rate: this could lead to blocking situation where there is still a hole to fill in a triplet but no available node satisfies the zone separation constraint. To circumvent this issue, we propose an algorithm based on finding cycles in a graph encoding of the assignment. As in section 2.1, we can explore the neigbours in a random order in the graph algorithms, to spread the triplets distribution.

A) Minimizing the zone discrepancy

First, notice that, given an assignment of partitions to *zones*, it is easy to deduce an assignment to *nodes* that minimizes the number of transfers for this zone assignment: For every zone z and every node $v \in z$, pick in any way a set P_v of partitions that where assigned to v in T', to z_v in T, with the cardinality of P_v smaller than n_v. Once all these sets are chosen, complement the assignment to reach the right utilization for every node. If $\#P_v > n_v$, it means that all the partitions that could stay in v (i.e. that were already in v and are still assigned to its zone) do stay in v. If $\#P_v = n_v$, then n_v partitions stay in v, which is the number of partitions that need to be in v in the end. In both cases, we could not hope for better given the partition to zone assignment.

Our goal now is to find a assignment of partitions to zones that minimizes the number of zone transfers. To do so we are going to represent an assignment as a graph.

Let $G_T = (X, E_T)$ be the directed weighted graph with vertices $(\mathbf{x}_i)_{1\leq i\leq N}$ and $(\mathbf{y}_z)_{z\in Z}$. For any $1 \leq i \leq N$ and $z \in Z$, E_T contains the arc:

- $(\mathbf{x}_i, \mathbf{y}_z, +1)$, if z appears in T'_i and T_i;
- $(\mathbf{x}_i, \mathbf{y}_z, -1)$, if z appears in T_i but not in T'_i;
- $(\mathbf{y}_z, \mathbf{x}_i, -1)$, if z appears in T'_i but not in T_i;
- $(\mathbf{y}_z, \mathbf{x}_i, +1)$, if z does not appear in T'_i nor in T_i.

In other words, the orientation of the arc encodes whether partition i is stored in zone z in the assignment T and the weight ± 1 encodes whether this corresponds to what happens in the assignment T'.

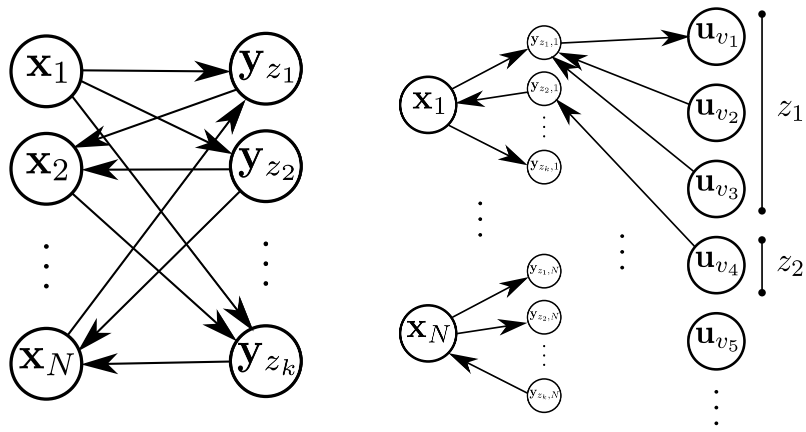

Figure 3: On the left: the graph G_T encoding an assignment to minimize the zone discrepancy. On the right: the graph G_T encoding an assignment to minimize the node discrepancy.

Notice that at every partition, there are three outgoing arcs, and at every zone, there are n_z incoming arcs. Moreover, if $w(e)$ is the weight of an arc e, define the weight of G_T by

$$\begin{aligned} w(G_T) := \sum_{e\in E} w(e) &= \#Z \times N - 4 \sum_{1\leq i\leq N} \#\{z \in Z \mid z \cap T_i = \emptyset, z \cap T_i' \neq \emptyset\} \\ &= \#Z \times N - 4 \sum_{1\leq i\leq N} 3 - \#\{z \in Z \mid z \cap T_i \neq \emptyset, z \cap T_i' \neq \emptyset\} \\ &= (\#Z - 12)N + 4Q_Z. \end{aligned}$$

Hence maximizing Q_Z is equivalent to maximizing $w(G_T)$.

Assume that their exist some assignment T^* with the same utilization $(n_v)_{v\in V}$. Define G_{T^*} similarly and consider the set $E_{\text{Diff}} = E_T \setminus E_{T^*}$ of arcs that appear only in G_T. Since all vertices have the same number of incoming arcs in G_T and G_{T^*}, the vertices of the graph (X, E_{Diff}) must all have the same number number of incoming and outgoing arrows. So E_{Diff} can be expressed as a union of disjoint cycles. Moreover, the edges of E_{Diff} must appear in E_{T^*} with reversed orientation and opposite weight. Hence, we have

$$w(G_T) - w(G_{T^*}) = 2 \sum_{e\in E_{\text{Diff}}} w(e).$$

Hence, if T is not optimal, there exists some T^* with $w(G_T) < w(G_{T^*})$, and by the considerations above, there must exist a cycle in E_{Diff}, and hence in G_T, with negative weight. If we reverse the edges and weights along this cycle, we obtain some graph. Since we did not change the incoming degree of any vertex, this is the graph encoding of some valid assignment T^+ such that $w(G_{T^+}) > w(G_T)$. We can iterate this operation until there is no other assignment T^* with larger weight, that is until we obtain an optimal assignment.

B) Minimizing the node discrepancy

We will follow an approach similar to the one where we minimize the zone discrepancy. Here we will directly obtain a node assignment from a graph encoding.

Let $G_T = (X, E_T)$ be the directed weighted graph with vertices $(\mathbf{x}_i)_{1\leq i\leq N}$, $(\mathbf{y}_{z,i})_{z\in Z, 1\leq i\leq N}$ and $(\mathbf{u}_v)_{v\in V}$. For any $1 \leq i \leq N$ and $z \in Z$, E_T contains the arc:

- $(\mathbf{x}_i, \mathbf{y}_{z,i}, 0)$, if z appears in T_i;
- $(\mathbf{y}_{z,i}, \mathbf{x}_i, 0)$, if z does not appear in T_i.

For any $1 \leq i \leq N$ and $v \in V$, E_T contains the arc:

- $(\mathbf{y}_{z_v,i}, \mathbf{u}_v, +1)$, if v appears in T'_i and T_i;
- $(\mathbf{y}_{z_v,i}, \mathbf{u}_v, -1)$, if v appears in T_i but not in T'_i;
- $(\mathbf{u}_v, \mathbf{y}_{z_v,i}, -1)$, if v appears in T'_i but not in T_i;
- $(\mathbf{u}_v, \mathbf{y}_{z_v,i}, +1)$, if v does not appear in T'_i nor in T_i.

Every vertex $\curvearrowleft_i$ has outgoing degree 3, every vertex $\mathbf{y}_{z,v}$ has outgoing degree 1, and every vertex $\mathbf{u}_v$ has incoming degree n_v. Remark that any graph respecting these degree constraints is the encoding of a valid assignment with utilizations $(n_v)_{v\in V}$, in particular no partition is stored in two nodes of the same zone.

We define $w(G_T)$ similarly:

$$\begin{aligned} w(G_T) := \sum_{e\in E_T} w(e) &= \#V \times N - 4 \sum_{1\leq i\leq N} 3 - \#(T_i \cap T'_i) \\ &= (\#V - 12)N + 4Q_V. \end{aligned}$$

Exactly like in the previous section, the existence of an assignment with larger weight implies the existence of a negatively weighted cycle in G_T. Reversing this cycle gives us the encoding of a valid assignment with a larger weight. Iterating this operation yields an optimal assignment.

C) Linear combination of both criteria

In the graph G_T defined in the previous section, instead of having weights 0 and ± 1, we could be having weights $\pm\alpha$ between $\mathbf{x}$ and $\mathbf{y}$ vertices, and weights $\pm\beta$ between $\mathbf{y}$ and $\mathbf{u}$ vertices, for some $\alpha, \beta > 0$ (we have positive weight if the assignment corresponds to T' and negative otherwise). Then

$$\begin{aligned} w(G_T) = \sum_{e\in E_T} w(e) &= \alpha\big((\#Z - 12)N + 4Q_Z\big) + \beta\big((\#V - 12)N + 4Q_V\big) \\ &= \mathrm{const} + 4(\alpha Q_Z + \beta Q_V). \end{aligned}$$

So maximizing the weight of such graph encoding would be equivalent to maximizing a linear combination of Q_Z and Q_V.

2.3 Algorithm

We give a high level description of the algorithm to compute an optimal 3-strict assignment. The operations appearing at lines 1,2,4 are respectively described by Algorithms 2,3 and 4.

Algorithm 1 Optimal 3-strict assignment

1: **function** OPTIMAL 3-STRICT ASSIGNMENT(N, $(c_v)_{v \in V}$, T')
2: $\quad$ $(n_v)_{v \in V} \leftarrow$ COMPUTE OPTIMAL UTILIZATION(N, $(c_v)_{v \in V}$)
3: $\quad$ $(T_i)_{1 \leq i \leq N} \leftarrow$ COMPUTE CANDIDATE ASSIGNMENT(N, $(n_v)_{v \in V}$)
4: $\quad$ **if** there was a previous assignment T' **then**
5: $\quad\quad$ $T \leftarrow$ MINIMIZATION OF TRANSFERS($(T_i)_{1 \leq i \leq N}$, $(T'_i)_{1 \leq i \leq N}$)
6: $\quad$ **end if**
7: $\quad$ **return** T.
8: **end function**

We give some considerations of worst case complexity for these algorithms. In the following, we assume $N > \#V > \#Z$. The complexity of Algorithm 1 is $O(N^3 \#Z)$ if we assume (H3A) and $O(N^3 \#Z \#V)$ if we assume (H3B).

Algorithm 2 can be implemented with complexity $O(\#V^2)$. The complexity of the function call at line 2 is $O(\#V)$. The difference between the sum of the subutilizations and $3N$ is at most the sum of the rounding errors when computing the $\hat{n}_v$. Hence it is bounded by $\#V$ and the loop at line 3 is iterated at most $\#V$ times. Finding the minimizing v at line 4 takes $O(\#V)$ operations (naively, we could also use a heap).

Algorithm 3 can be implemented with complexity $O(N^3 \times \#Z)$. The flow graph has $O(N + \#Z)$ vertices and $O(N \times \#Z)$ edges. Dinic's algorithm has complexity $O(\#\text{Vertices}^2 \#\text{Edges})$ hence in our case it is $O(N^3 \times \#Z)$.

Algorithm 4 can be implented with complexity $O(N^3 \#Z)$ under (H3A) and $O(N^3 \#Z \#V)$ under (H3B). The graph G_T has $O(N)$ vertices and $O(N \times \#Z)$ edges under assumption (H3A) and respectively $O(N \times \#Z)$ vertices and $O(N \times \#V)$ edges under assumption (H3B). The loop at line 3 is iterated at most N times since the distance between T and T' decreases at every iteration. Bellman-Ford algorithm has complexity $O(\#\text{Vertices}\#\text{Edges})$, which in our case amounts to $O(N^2 \#Z)$ under (H3A) and $O(N^2 \#Z \#V)$ under (H3B).

3 TODO

- reunion deux fleurs : autres modes, autres contraintes

Algorithm 2 Computation of the optimal utilization

function COMPUTE OPTIMAL UTILIZATION(N, $(c_v)_{v \in V}$)
    $(\hat{n}_v)_{v \in V} \leftarrow$ COMPUTE SUBUTILIZATION(N, $(c_v)_{v \in V}$)
    while $\sum_{v \in V} \hat{n}_v < 3N$ **do**
        Pick $v \in V$ minimizing $\frac{c_v}{\hat{n}_v + 1}$ and such that $\sum_{v' \in z_v} \hat{n}_{v'} < N$
        $\hat{n}_v \leftarrow \hat{n}_v + 1$
    end while
    return $(\hat{n}_v)_{v \in V}$
end function

function COMPUTE SUBUTILIZATION(N, $(c_v)_{v \in V}$)
    $R \leftarrow 3$
    for $v \in V$ **do**
        $\hat{n}_v \leftarrow$ unset
    end for
    for $z \in Z$ **do**
        $c_z \leftarrow \sum_{v \in z} c_v$
    end for
    $C \leftarrow \sum_{z \in Z} c_z$
    while $\exists z \in Z$ such that $R \times c_z > C$ **do**
        for $v \in z$ **do**
            $\hat{n}_v \leftarrow \left\lfloor \frac{c_v}{c_z} N \right\rfloor$
        end for
        $C \leftarrow C - c_z$
        $R \leftarrow R - 1$
    end while
    for $v \in V$ **do**
        if $\hat{n}_v =$ unset **then**
            $\hat{n}_v \leftarrow \left\lfloor \frac{R c_v}{C} N \right\rfloor$
        end if
    end for
    return $(\hat{n}_v)_{v \in V}$
end function

Algorithm 3 Computation of a candidate assignment

function COMPUTE CANDIDATE ASSIGNMENT(N, $(n_v)_{v \in V}$)
    Compute the flow graph G
    Compute the maximal flow f using Dinic's algorithm with randomized neighbours enumeration
    Construct the assignment $(T_i)_{1 \leq i \leq N}$ from f
    return $(T_i)_{1 \leq i \leq N}$
end function

Algorithm 4 Minimization of the number of transfers

```
function MINIMIZATION OF TRANSFERS((T_i)_{1≤i≤N}, (T'_i)_{1≤i≤N})
    Construct the graph encoding G_T
    repeat
        Find a negative cycle γ using Bellman-Ford algorithm on G_T
        Reverse the orientations and weights of edges in γ
    until no negative cycle is found
    Update (T_i)_{1≤i≤N} from G_T
    return (T_i)_{1≤i≤N}
end function
```