Theoretical Computer Science 566 (2015) 12–25

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The fault-tolerant capacitated K-center problem

Shiri Chechik [a,*], David Peleg [b,1]

[a] Department of Computer Science, Tel-Aviv University, Israel
[b] Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel

ARTICLE INFO

Article history:
Received 18 June 2014
Received in revised form 7 November 2014
Accepted 17 November 2014
Available online 24 November 2014
Communicated by G. Ausiello

Keywords:
K-center
Approximation algorithms
Fault-tolerance

ABSTRACT

The *capacitated K-center (CKC)* problem calls for locating K service centers in the vertices of a given weighted graph, and assigning each vertex as a client to one of the centers, where each service center has a limited service capacity and thus may be assigned at most L clients, so as to minimize the maximum distance from a vertex to its assigned service center. This paper studies the fault-tolerant version of this problem, where one or more service centers might fail simultaneously. We consider two variants of the problem. The first is the *α-fault-tolerant capacitated K-center* (α-FT-CKC) problem. In this version, after the failure of some centers, all nodes are allowed to be reassigned to alternate centers. The more conservative version of this problem, hereafter referred to as the *α-fault-tolerant conservative capacitated K-center* (α-FT-CKC) problem, is similar to the α-FT-CKC problem, except that after the failure of some centers, only the nodes that were assigned to those centers before the failure are allowed to be reassigned to other centers. We present polynomial time algorithms that yield 9-approximation for the α-FT-CKC problem and 17-approximation for the α-FT-CKC problem.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Problems and results

The *basic K-center* problem is defined as follows. For a given weighted graph G, it is required to select a set S of up to K nodes that will host service centers, and to assign each vertex as a client to a service center, so as to minimize the maximum distance from a vertex to its assigned service center. Formally, the objective function is

$$\min_{S\subseteq V,|S|\leq K}\left\{\max_{v\in V}\{\delta_G(v,S)\}\right\},$$

where $\delta_G(v,S)$ is the distance in G from v to its closest node in S.

This paper considers the *capacitated K-center (CKC)* problem, where it is required to locate K service centers in a weighted graph, and to assign each of the vertices to one of the service centers, where each service center has a limited service capacity and may be assigned at most L vertices, so as to minimize the maximum distance from a vertex to its assigned

* Corresponding author.
E-mail addresses: shiri.chechik@gmail.com (S. Chechik), david.peleg@weizmann.ac.il (D. Peleg).

[1] Supported in part by the Israel Science Foundation (grant 894/09), the United States–Israel Binational Science Foundation (grant 2008348), the Israel Ministry of Science and Technology (infrastructures grant 3-6478) and the Citi Foundation, Citigroup.

http://dx.doi.org/10.1016/j.tcs.2014.11.017
0304-3975/© 2014 Elsevier B.V. All rights reserved.

service center. All nodes serve as clients, namely, it is required to assign each node to a center even though the node may contain a center place on it.

In the fault-tolerant version of this problem, one or more service centers might fail simultaneously. After the failure of the service centers, it is still required to assign each node to some surviving center, obeying the constraint that each center can still serve at most L nodes. The objective is to minimize the maximum distance from a node to its assigned center, under all possible subsets of up to α failed service centers, for some integer α.

We consider two variants of the problem. The first is the *α-fault-tolerant capacitated K-Center (α-FT-CKC)* problem. In this version, after the failure of some centers, all nodes are allowed to be reassigned to other centers.

The second variant is a more conservative version of this problem, hereafter referred to as the *α-fault-tolerant conservative capacitated K-center (α-FT-CCKC)* problem, which is similar to the α-FT-CKC problem, except that after the failure of some centers, only the nodes that were assigned to those centers before the failure are allowed to be reassigned to other centers. All other nodes continue to be served by their original centers. We present a polynomial time algorithms that yields 9-approximation for the α-FT-CKC problem and 17-approximation for the α-FT-CCKC problem.

Our definition assumes that a failed node can no longer host a center and supply demands, but its own demands must still be supplied. Notice that in all of the fault-tolerant problems mentioned above, the capacity L must be greater than 1, since if $L = 1$, then all nodes in the graph must be allocated as centers, and if one or more of the nodes fail, then there is not enough overall capacity to handle all nodes.

1.2. Related work

The basic K-center problem is known to be NP-hard [7] and admits a 2-approximation algorithm [8–12].

Some generalizations of the K-center problem were considered in the literature (e.g. [3,5,11,15]). One generalization for the K-center problem, referred to as the *capacitated K-center* problem, was introduced in [2]. In this version of the problem it is required to locate K service centers in a graph, and to assign each of the vertices to one of the service centers, where each service center has a limited capacity and may be assigned at most L vertices, so as to minimize the maximum distance from a vertex to its assigned service center. An approximation algorithm with ratio 10 was presented in [2] for this problem. This approximation ratio was later improved by Khuller and Sussmann [14] to 6, or to 5 in the version where a single node is allowed to host several service centers.

Cygan, Hajiaghayi and Khuller [4] considered the more general variant of the capacitated K-center problem, where instead of uniform capacity function we have a general function. They obtained an algorithm with a constant approximation ratio for this version. This approximation ratio was later improved by An et al. [1] to 9.

The basic K-center problem was considered in a failure-prone setting in [13]. In this version it is again required to create K service centers, but some of these centers may fail. After the failure, each node is assigned to the closest surviving center. Here, too, the objective is to minimize the maximum distance from a node to its assigned service center. The problem is parameterized by an integer parameter α, bounding the maximum number of centers that may fail in the worst case. In this version, where each center has an unlimited capacity, the problem can be given the following alternative formulation. Each node is assigned to $\alpha + 1$ service centers and the objective is to minimize the maximum distance from a node to any of its $\alpha + 1$ service centers. Two subversions of this problem were considered in [13]. In the first subversion, every node is required to have $\alpha + 1$ centers close to it, whereas in the second subversion, this requirement is applied only to a node that does not have a center placed on it. For the first subversion, a 3-approximation algorithm is given for any α and a 2-approximation algorithm for $\alpha < 3$. For the second subversion, a 2-approximation ratio algorithm is given for any α. Observe that in the capacitated version of the problem, which is studied here, it is harder to manage α-fault-tolerance, since it is not enough to make sure that each node has $\alpha + 1$ service centers close to it; the difficulty is that it could be the case that the nearby service centers do not have enough free capacity to handle this node.

2. Preliminaries

A solution for the capacitated K-center problem, in both the failure free and failure-prone settings, is a set R of up to K nodes in which centers are to be located. Once the set of locations R is determined, it is then required to assign every node v (as a client) to a service center $ctr(v)$ from R in such a way that the number of nodes served by each center is at most the capacity L. Formally, letting $dom(r)$ denote the set of clients served by a center r, $dom(r) = \{v \mid ctr(v) = r\}$, it is required that $|dom(r)| \leq L$ for every $r \in R$. Such an assignment is termed a *feasible* assignment. A solution R is feasible if it has sufficiently many centers to handle the capacity of all nodes, namely, $K \geq \lceil n/L \rceil + \alpha$. The cost of a solution R is the maximum distance from a node to its assigned center.

Note that in the failure free setting, given a solution R, one can efficiently find an optimal feasible assignment, namely, an assignment for each node to a center of R, that satisfies the constraint that each center serves at most L nodes, and minimizes the cost. This can be done using the following bottleneck method. Sort all edge weights in nondecreasing order, let the sorted list of edges be $e_1, \ldots, e_{|E|}$. We assume G is a complete weighted graph (a non-complete graph G can be made

complete by defining the weight of each edge (x, y) as the length of the shortest path between x and y in G). Note that the cost of every feasible solution R is equal to $\omega(e_i)$ for some $1 \leq i \leq m$. For each weight $W = \omega(e_i)$, define the graph G_W to be the subgraph obtained from G by taking only edges of weight at most W. Consider each possible value of W from $\omega(e_1)$ to $\omega(e_{|E|})$. For each W, the goal is to check if there is a way to assign all nodes to R using only edges from G_W, under the constraint that each center in R can serve at most L nodes. This can be done by defining a suitable flow problem and finding the max flow on it. Construct a bipartite graph $\tilde{G} = (R, V, E')$ where for every $r \in R$ and $v \in V$, there is an edge (r, v) in E' iff the distance from r to v is at most W. Set the capacity of these edges to 1. Add two auxiliary nodes s and t, connect s to each node in R with an edge of capacity L and add an edge of capacity 1 from each node of V to t. If the maximum (s, t)-flow on $\tilde{G}$ is $|V|$, then the K center problem has an assignment of cost W (where a client v is assigned to a center r if the edge (v, r) is used in the flow).

Given a solution R, denote by $\rho(R)$ its cost, namely, the minimum W such that there is a way to assign all nodes to R using only edges from G_W, under the capacity constraint.

In the failure-prone setting, given a solution R of the capacitated K-Center problem and a set of failed service centers F, denote by $\rho(R, F)$ the minimum weight W such that there is a way to assign all nodes to centers in $R \setminus F$ using only edges from G_W, under the constraint that each center in $R \setminus F$ can serve at most L clients, namely, $\rho(R, F) = \rho(R \setminus F)$. Denote the worst case service radius by

$$\rho_\alpha(R) = \max_{|F| \leq \alpha} \{\rho(R, F)\}.$$

The α-FT-CKC problem requires finding a solution R that minimizes $\rho_\alpha(R)$. In contrast, in the α-FT-CCKC problem the cost of the optimal solution might be higher than $\rho_\alpha(R)$, due to the additional constraint that only the nodes that were served by the set of failed centers F can be reassigned.

3. High level structure of the algorithm

Our algorithms for both versions are based on using the (non-fault-tolerant) algorithms proposed by Khuller and Sussmann [14] as a starting point, and introducing the necessary modifications to make them tolerant against the failure of some centers. For the sake of completeness we first describe the main ideas of [14] (which in turn follow [9,2]).

As in [6,9–11,14], the algorithms we present follow the general strategy of the bottleneck method. Turn G into a complete graph as described in Section 2, and sort all edge weights in nondecreasing order; let the sorted list of edges be $e_1, \ldots, e_{|E|}$. For each weight W, define the graph G_W to be the subgraph of G containing only edges e_i of weight $\omega(e_i) \leq W$. Run algorithm Main(G_W, K, L) for each value W from $\omega(e_1)$ to $\omega(e_{|E|})$, until a feasible solution (with K or fewer centers) is obtained. (Note that instead of sequentially iterating on $w(e_i)$ values to find a feasible solution, one can rather invoke a binary search for improving the running time.) In each iteration, consider the subgraph G_W and treat it as an unweighted graph. In this graph, define the distance $dist(u, v, G_W)$ between two nodes as the number of edges on the shortest path between them. For the weight W, let K^*_W denote the minimal number of centers needed for a feasible solution of cost at most W. Algorithm Main finds a solution for the problem on G_W using some number K_W of centers. We prove in Lemma 4.5 and Corollary 5.4 that if $K_W > K$, then $K^*_W > K$, i.e., there is no feasible solution of cost at most W using at most K centers.

For a node $v \in V$, let $\Gamma_i(v) = \{u \mid dist(u, v, G_W) \leq i\}$, and $N_i(v) = \{u \mid dist(u, v, G_W) = i\}$. The algorithms presented in [14] use three central procedures. The first procedure, referred to as Select_Monarchs, first constructs the *power graph* G^2_W, obtained from G_W by adding edges between all pairs that have a common neighbor, and then selects an independent set M in G^2_W and places centers on them. The nodes of M are referred to as *monarchs*.

The set M of monarchs is selected by the following iterative process. Initially, all nodes are unmarked. After choosing a new monarch m, mark all unmarked nodes at distance (in G_W) 1 or 2 from it. The set of new marked nodes is referred to as the *empire* of m, denoted $Emp(m)$. In each iteration we choose a new monarch m' by picking an unmarked node m' that is adjacent to a marked node v. This v is termed the *deputy* of m', and v's monarch, namely, the monarch m such that $v \in Emp(m)$, is termed the *parent* monarch of m' and denoted $Parent(m')$. Note that v must be at distance 2 from its monarch, and in addition, m' is the only monarch at distance 1 from v. For a monarch m, $Emp(m) \subseteq \Gamma_2(m)$ in G_W. This procedure yields a rooted tree T of monarchs with the property that the distance between a monarch and its parent monarch in G_W is three.

A second procedure of [14], referred to as Assign_Domains, tries to assign to each monarch a domain of L nodes at distance at most 2 from it in G_W. A monarch may be assigned vertices from the empires of other monarchs, in case no more nodes from its empire are unassigned.

The third Procedure, ReAssign, handles all unassigned nodes. This procedure creates new centers and assigns all unassigned nodes to centers. The centers are chosen for each connected component separately; this can be done as no nodes will be assigned to centers in different connected components of G_W assuming the optimal cost is W.

The main algorithm is given in Procedure Main. This main procedure is used also for all algorithms presented in this paper. However, the internal components need to be modified for each version of the problem.

Algorithm Main($G = (V, E), K, L$)

1. for every edge weight W in non-decreasing order of weights do:
   - let $G_W = (V, E_W)$ where $E_W = (e \in E \mid \omega(e) \leq W)$
   - unmark all vertices
   - if Assign_Centers(G_W) then exit

Algorithm Assign_ Centers(G_W)

1. Suppose G_W consists of connected components $G_W^1, ..., G_W^\ell$
2. for $1 \leq c \leq \ell$, let n_W^c = number of nodes in connected component G_W^c
3. let $K_W \leftarrow \sum_c \lceil n_W^c / L \rceil$
4. if $K_W > K$ then return false
5. for each connected component G_W^c do:
   - Select_Monarchs(G_W^c)
   - Assign_Domains(G_W^c)
   - ReAssign(G_W^c)
6. if the total number of centers used is more than K then return false
7. return true

4. Constant approximation for the α-FT-CKC problem

In this section we show a constant approximation algorithm for the α-FT-CKC problem. Recall that in this version of the problem, after the failure of a set of centers F, all nodes can be reassigned again.

In this algorithm, we modify procedures Select_ Monarchs, Assign_Domains and ReAssign of [14], and in addition introduce another procedure, named ReAssign_by_F, which is invoked after the failure of a set of centers F.

Lemma 4.1. *If the α-FT-CKC problem has a solution of cost at most $W = w(e_i)$ for some $1 \leq i \leq m$, then the minimum node degree in G_W is at least α. Moreover, every solution of cost at most W requires having $\alpha + 1$ service centers in $\Gamma_1(v)$ for every $v \in V$.*

Proof. Assume otherwise, and consider some node v of degree less than α or with less than $\alpha + 1$ service centers in $\Gamma_1(v)$. Then in a failure event where all centers located in $\Gamma_1(v)$ fail simultaneously, v is forced to be served by a center at distance more than W, contradicting the feasibility of the solution. □

By Lemma 4.1, we restrict ourself to the case where the minimum node degree in G_W is at least α. Our Procedure Select_Monarchs first finds a set M_1 of monarchs, which is a maximal independent set in G_W^2, in a way similar to Procedure Select_Monarchs of [14] (as described in Section 3). We call these monarchs *major* monarchs. The algorithm of [14] required a single type of monarchs. In contrast, we introduce a second type of monarchs, referred to as *minor* monarchs, which are selected as follows. For every major monarch m, choose a set of $\alpha - 1$ neighbors of m, not including the deputy of m. (Note that the degree of m is at least α, hence such $\alpha - 1$ neighbors must exist.) Denote this set by $minors(m)$, and set $major(m') = m$ for each $m' \in minors(m) \cup \{m\}$. For every major monarch m, denote the set $\{m\} \cup minors(m)$ by $team(m)$. Let M_2 be the set of all minor monarchs. As claimed above, the optimal solution must contain at least $\alpha + 1$ service centers in $\Gamma_1(m)$, therefore at least α of m's neighbors must host centers. The reason for choosing just $\alpha - 1$ neighbors, instead of α, is that we would like to keep the deputy of m available for placing a center on it in Procedure ReAssign, and setting centers at only $\alpha - 1$ neighbors is sufficient for our needs, as will become clearer later on. In Procedure Select_Monarchs, each time a new major monarch m is selected, all unmarked nodes at distance at most 2 from m are marked and assigned to the empire $Emp(m)$ of m.

We would like to serve as many nodes as possible using the monarchs of $M = M_1 \cup M_2$, where every monarch m can serve clients from $\Gamma_2(major(m))$. Procedure Assign_Domains solves this problem by constructing a suitable graph $\tilde{G}$ and finding the minimum cost maximum flow on it. This procedure constructs a bipartite graph $\tilde{G} = (M, V, E')$, where M is the set of all monarchs, V is the set of nodes of the graph and E' contains an edge (m, v) for every monarch m and node v in $\Gamma_2(major(m))$. The goal now is to assign as many nodes as possible to the service centers, where a node v can be assigned to a center m if it has an edge to it in $\tilde{G}$, i.e., $(v, m) \in E'$. Add to $\tilde{G}$ new nodes s and t and edges $\{(s, m) \mid m \in M\}$ and $\{(v, t) \mid v \in V\}$. For every monarch $m \in M$ set a capacity of $u(s, m) = L$ and for every node $v \in V$ set a capacity of $u(m, v) = 1$. Set the cost of edge (m, v) to be $c(m, v) = 0$ if $m = v$ and 1 otherwise. Compute a min-cost maximum integral flow in this new graph. Set $dom(m)$ to be all nodes that got a

unit flow from m. For a set of monarchs $X \subseteq M$, let $dom(X) = \bigcup_{m \in X} dom(m)$. We note that, unlike in [14], here the monarchs can be at distance 1 from one another. Therefore, the monarchs can serve one another, we hence do not assume that all monarchs serve themselves (it could be that by forcing monarchs to serve themselves we will hurt the maximum flow). However, by setting the cost of the edge from a monarch to itself in the graph $\tilde{G}$ to be 0 (and the rest of the edges 1), we make sure that even if a monarch does not serve itself it is still served by another monarch (otherwise we can get a lower cost solution with the same maximum flow). In other words, all monarchs are served.

We now turn to describe Procedure ReAssign. For every major monarch m, let $unassigned(m)$ be the set of all nodes in its empire $Emp(m)$ that are unassigned, namely, that do not belong to the domain of any monarch. Consider the tree T of major monarchs as described in Section 3. The algorithm assigns clients to the monarchs in a bottom–up process on T. More precisely, the process maintains a copy T' of T consisting of all the monarchs that have not been handled yet. In each step of the algorithm, pick a leaf m from the tree T' for processing and remove it from T'. Let $k'L + \epsilon$ be the number of nodes in $unassigned(m)$ plus the number of unassigned nodes passed to m by its children monarchs in T. Allocate k' new centers at free nodes in m's empire and assign the remaining ϵ unassigned nodes to the monarch m, releasing at most ϵ nodes from m's domain, and pass these nodes to m's parent in T for reassignment (we say that a node is free if no center is placed on it). If m is the root monarch of T, then allocate enough new centers to serve all $k'L + \epsilon$ unassigned nodes. For each of these new allocated centers, we say that m is their major monarch. Notice that if up to α failures might occur, then any feasible solution must contain at least $\lceil n/L \rceil + \alpha$ centers, as otherwise, after the failure of any α centers, there is not enough capacity left to handle all nodes. So in the last step of Procedure ReAssign, check if the algorithm creates fewer than $\lceil n/L \rceil + \alpha$ centers; if so, then add new centers to complete to $\lceil n/L \rceil + \alpha$.

Algorithm Select_ Monarchs(G_W)

1. pick an arbitrary vertex $v \in G_W$ and set $Q = \{v\}$
2. $Parent(v) \leftarrow nil$
3. while Q has unmarked nodes do:
   - remove an unmarked node v from Q
   - make v a monarch, mark it and add it to M_1
   - add v to T, set its parent in T to be $Parent(v)$
   - for all $u \in \Gamma_2(v)$
     - if u is unmarked then add u to $Emp(v)$ and mark u
   - for all $u \in Emp(v) \cap N_2(v)$
     - for all $w \in N_1(u)$
       - if w is unmarked and $w \notin Q$ then set $Parent(w) = v$, $deputy(w) \leftarrow u$ and add w to Q
4. $M_2 \leftarrow \emptyset$
5. for each $m \in M_1$ do:
   - $minors(m) \leftarrow \{\alpha - 1$ arbitrary nodes from $N_1(m) \setminus \{deputy(m)\}\}$
   - $M_2 \leftarrow M_2 \cup minors(m)$
   - $major(m') \leftarrow m$ for each $m' \in minors(m) \cup \{m\}$
6. let $M \leftarrow M_1 \cup M_2$

A *light* monarch is one whose domain is of size less than L. Let K_L denote the number of light monarchs and n_L be the number of vertices belonging to the domains of light monarchs, and let n be the total number of vertices.

Define the set of *useful monarchs* U as follows. Let U_0 be the set of light monarchs. Increase U_0 by an iterative process, in which U_j is created by U_{j-1} by adding to it any monarch that contains in its domain a node that could have been assigned to a node in U_{j-1}. Formally, let

$$U_j = U_{j-1} \cup \Big\{ m \in M \;\Big|\; \exists v \in dom(m), \exists m' \in U_{j-1} \text{ and } dist\big(v, major\big(m'\big)\big) \leq 2 \text{ in } G_W \Big\}.$$

Let U be the largest set obtained in this way. We say that a monarch m is *overloaded* if there are unassigned nodes in $\Gamma_2(major(m))$.

We note that the definition of overloaded monarchs is close to the definition of heavy monarchs introduced in [14]. The difference between the two terms is that an overloaded monarch m has unassigned vertices in $\Gamma_2(major(m))$ instead of in its empire. In addition, note that the definition of U is similar to the one presented in [14] with the slight modification that

we use $dist(v, major(m'))$ instead of $dist(v, m')$. The reason for these modifications is that in our Procedure Assign_Domains, every monarch m' can serve nodes from $\Gamma_2(major(m'))$ rather than from $\Gamma_2(m')$.

Algorithm Assign_ Domains(G_W)

1. let $M \leftarrow M_1 \cup M_2$ /* the set of monarchs in G_W */
2. let $E' \leftarrow \{(m, v) \mid v \in \Gamma_2(major(m))\}$
3. construct a bipartite graph $\tilde{G} = (M, V, E')$
4. add vertices s and t and edges $\{(s, m) \mid m \in M\}$ and $\{(v, t) \mid v \in V\}$
5. for $m \in M, v \in V$, set capacities $u(s, m) = L, u(m, v) = 1$ and $u(v, t) = 1$. Cost of edge $c(m, v) = 0$ if $v = m$ and 1 otherwise
6. compute a min-cost maximum integral flow $MCMF(\tilde{G})$ in $\tilde{G}$
7. for each monarch m, set $dom(m) = \{v \mid v$ receives one unit of flow from m in $\tilde{G}\}$

The following lemmas summarize some basic properties.

Lemma 4.2. *The set U does not contain any overloaded monarchs.*

Proof. Assume, towards contradiction, that an overloaded monarch m is added to U_j at iteration j. It is possible to transfer a node that m serves to a monarch in U_{j-1}, releasing another node v' that can be transferred to a monarch in U_{j-2}, and this process can be continued until reaching a monarch m_0 in U_0. The monarch m_0 is light, therefore it has at most $L - 1$ clients in its domain and can serve another node. The node m can now absorb an unassigned node in $\Gamma_2(major(m))$. Hence the process yields a higher flow, a contradiction. □

Lemma 4.3. *Consider some major monarch $m \in M_1$. If one of the nodes in $team(m)$ belongs to U, then all of them do.*

Proof. This follows trivially from the fact that all monarchs in $team(m)$ can serve the same constituency, namely $\Gamma_2(m)$ in G_W. □

Denote by $dom^*(\theta)$ the set of nodes that are served by some center θ before any failure occurs in the optimal solution.

Lemma 4.4. *Consider a monarch $m \in U \cap M_1$ and a center θ in an optimal solution such that $\theta \in \Gamma_1(m)$. Then $dom^*(\theta) \subseteq dom(U)$.*

Proof. Consider a monarch $m \in U \cap M_1$ and a center θ in an optimal solution such that $\theta \in \Gamma_1(m)$.

Suppose θ also serves some node u in the optimal solution, i.e., $u \in dom^*(\theta)$. If u does not belong to any domain (i.e., u is unassigned), then a contradiction is derived by Lemma 4.2 since it follows that the distance from u to m is at most 2, hence m is overloaded. Hence $u \in dom(m')$ for some m'. Then since $u \in \Gamma_2(m)$ and $m \in U$, also $m' \in U$, as required. □

Lemma 4.5. *In every solution of radius cost W, the number of centers required satisfies $K^*_W \geq \max\{K_L + \lceil (n - n_L)/L \rceil, \lceil n/L \rceil + \alpha\} = K_W$.*

Proof. By Lemma 4.1, at the neighborhood of each major monarch $m \in M_1$ where one of the nodes in $team(m)$ belongs to U there must be at least $\alpha + 1$ distinct centers. By Lemma 4.4, these centers cannot cover any node that is not in $dom(U)$. So the number of required centers is $K^*_W \geq |U| + \lceil (n - |dom(U)|)/L \rceil = K_L + \lceil (n - n_L)/L \rceil$, where the last equality comes from the fact that U contains all the light monarchs, plus monarchs that serve L nodes. In addition any feasible solution must contain at least $\lceil n/L \rceil + \alpha$ centers, as otherwise, after the failure of any α centers, there is not enough capacity left to handle all nodes. We get that $K^*_W \geq \max\{K_L + \lceil (n - n_L)/L \rceil, \lceil n/L \rceil + \alpha\}$. Procedure Select_Monarchs creates $K_L + (dom(M) - n_L)/L$ centers and Procedure ReAssign creates $\lceil n - dom(M)/L \rceil$ additional centers. In addition, Procedure ReAssign ensures that there are at least $\lceil n/L \rceil + \alpha$ allocated centers, by allocating additional centers in case fewer centers were allocated so far. Therefore, the total number of service centers created by the algorithm is $\max\{K_L + \lceil (n - n_L)/L \rceil, \lceil n/L \rceil + \alpha\}$. □

Procedure ReAssign is similar to the one given in [14]; for the sake of completeness we describe it here as well.

Algorithm ReAssign(G_W)

1. let $M = M_1 \cup M_2$ be the set of monarchs in G_W
2. for each monarch $m \in M_1$, set $unassigned(m) = (\{m\} \cup Emp(m)) \setminus dom(M)$
3. let T be the tree of monarchs M_1 in G_W
4. for each node m in T, set $passed(m) = \emptyset$
5. while T is not empty do
   - remove a leaf node m from T
   - let $|unassigned(m) + passed(m)| = k'L + \epsilon$
   - allocate k' new centers at free nodes in m's empire and assign $k'L$ of the nodes to them
   - assign the ϵ remaining nodes to monarch m, releasing up to ϵ nodes in $dom(m)$
   - add the released nodes to $passed(Parent(m))$, unless m is the root monarch and then allocate a new center in a free node in m's empire and assign to it the ϵ remaining nodes
   - delete m from T
6. Let M' be all centers allocated so far
7. If $|M'| < \lceil n/L \rceil + \alpha$, allocate $\lceil n/L \rceil + \alpha - |M'|$ new centers at arbitrary free nodes

Lemma 4.6. *Each major monarch $m \in M_1$ has sufficiently many free nodes in its empire to allocate centers for all the clients in $unassigned(m)$ and $passed(m)$.*

Proof. Each monarch passes at most $L - 1$ nodes to its parent. Each child monarch of monarch m has a unique deputy. In addition, all these deputies do not contain a center yet, so they are available to be allocated as centers. Let S be the set of all these deputies. We get that there are sufficiently many deputies to handle all clients in $passed(m)$ and in $S \cap unassigned(m)$. Next, we claim that all other nodes v in $unassigned(m)$ also do not contain a center placed on them yet, namely, $v \notin M'$. To see this, assume, towards contradiction, that there exists an unassigned node v such that $v \in M'$. Then we can replace some node that v' serves with v itself, yielding a max flow with a lower cost, a contradiction. We get that all $unassigned(m) \setminus S$ are available to be allocated as centers, so clearly can handle all nodes in $unassigned(m) \setminus S$ (as $L \geq 1$). □

We now turn to describe the main difference between the algorithm of [14] and ours, designed to handle also failures. When a set F of at most α centers fails, each of the clients of $dom(F)$ needs to be reassigned to one of the surviving centers. Hence after the failure of a set of centers F, we invoke the procedure ReAssign_by_F, which handles the reassignment, possibly by performing a sequence of transfers until reaching a "free spot", i.e., a surviving center that currently serves fewer than L clients. First note that as we make sure that there are at least $\lceil n/L \rceil + \alpha$ centers, the capacity of all surviving centers is sufficient to serve all nodes. We note that given a set of centers R and a set of failures F, one can efficiently find an optimal feasible assignment using the centers $R \setminus F$ that minimizes the cost (as mentioned above for the failure-free case). Hence Procedure ReAssign_by_F is used only for the analysis, and specifically, for bounding the approximation ratio. The idea of Procedure ReAssign_by_F is to assign each node $x \in dom(F)$ to a "free spot" by a sequence of transfers that reassign x to some other center, possibly causing another node to become unassigned, then looking for a center for this client, and so on, until reaching the "free spot".

More precisely, Procedure ReAssign_by_F is as follows. Let $F = \{f_1, ..., f_k\}$, for some $k \leq \alpha$, be the set of failed centers. Think of every surviving center r as having $|dom(r)|$ "occupied" capacity and $f(r) = L - |dom(r)|$ "free" places for hosting new clients. Consider all nodes of $dom(F)$ and assign for each unassigned node v a unique free place in a non-faulty center. Let $free(v)$ be that center. Note that at most $f(r)$ nodes can be assigned to the same center r if it contains $f(r)$ free places, namely it serves $L - f(r)$ nodes. The idea now is to perform a sequence of transfers from each unassigned node v until reaching the free place in $free(v)$. For each unassigned node $v \in dom(f)$ for some $f \in F$, find the shortest *monarch path* $MP(v)$ in the tree of monarchs T between the major monarch of f and the major monarch of $free(v)$ (namely, the major monarch m such that $free(v) \in Emp(m)$). Let $MP(v) = (m_1, ..., m_j)$ be this shortest path, where m_1 is the major monarch of f and m_j is the major monarch of $free(v)$. Naively, we could apply the following process. Assign v to m_1, and to enable that, cancel the assignment of some other node v_1 in m_1, making it unassigned. Next, assign v_1 to m_2, making some other node v_2 unassigned, and so on, and by a sequence of transfers reach m_j and make some node v_j that was originally served by m_j unassigned and finally assign v_j to the center $free(v)$.

However, this naive rolling process does not necessarily yield a "good" solution, namely, one in which each node is assigned to a "relatively" close center. To see this, note that for all unassigned nodes v, the shortest monarch path $MP(v)$

in T from the major monarch of v to the major monarch of $free(v)$ could pass through some major monarch m. As m can serve only L nodes, it cannot serve all nodes passed to it during this process, so it must pass some of these nodes further. This could result in a large approximation ratio. Luckily, each major monarch has $\alpha - 1$ minor monarchs at distance 1 from it, which can be used in order to avoid such a situation. Select, for each failed center f and for each major monarch m, such that $f \notin team(m)$, a different non-faulty center from $team(m)$, and denote this center by $\chi(f, m)$.

Consider now an unassigned node v and let $f \in F$ be the center that serves v previous to the failure event. Let $MP(v) = (m_1, ..., m_j)$ be this shortest path in T from the major monarch of f to the major monarch of $free(v)$. Assign v to $\chi(f, m_2)$, thereby releasing some other node v_2 that was originally served by $\chi(f, m_2)$ and making it unassigned. Repeating this, one gets a sequence of transfers that reaches m_j and releases some node v_j that was originally served by $\chi(f, m_j)$. Finally, assign this node v_j to the center $free(v)$.

Algorithm ReAssign_by_F(G_W)

1. let $M = M_1 \cup M_2$ be the set of monarchs in G_W
2. let $F = \{f_1, ..., f_k\}$ for some $k \leq \alpha$ be the set of centers that fail
3. for each failed center f and for each monarch $m \in M_1$ such that $f \notin team(m)$ do:
   - select a different non-faulty center $r \in team(m)$
   - Set $\chi(f, m) \leftarrow r$
4. for each node v that was served by some $f \in F$ do:
   - assign v to a unique free place in a non-faulty center, set $free(v)$ to be the center with the free place
   - let the path in the tree of monarchs T from the major monarch of f to the major monarch of $free(v)$ be $MP = (m_1, ..., m_j)$, where m_j is the major monarch of $free(v)$
   - assign v to $\chi(f, m_2)$ releasing some other node from the original nodes $\chi(f, m_2)$ served, this node is assigned to $\chi(f, m_3)$ and so on until $\chi(f, m_j)$. The node that the server $\chi(f, m_j)$ released is then assign to $free(v)$ which can absorb the extra node

The following two lemmas establish the desired stretch bound. Let W be the first value for which Algorithm Assign_ Centers returns true.

Lemma 4.7. *Let c^* be the optimal solution to the fault-tolerant capacitated K-center problem, namely, the set of vertices such that $\rho_\alpha(c^*)$ is minimal. Then $\rho_\alpha(c^*) \geq W$.*

Proof. By Lemma 4.5, for every value W', $K^*_{W'} \geq K_{W'}$, where $K^*_{W'}$ is the number of centers needed in the optimal solution in order to get a cost of W' and $K_{W'}$ is the number of centers Algorithm Assign_Centers uses for W'. The algorithm stops once it finds a solution that uses at most K centers. Therefore, W is the lowest cost that can be attained using at most K centers. □

Lemma 4.8. *After the failure of a set F where $|F| \leq \alpha$, the reassignment process ensures that each client is assigned a center at distance at most 9 in G_W.*

Proof. Before the failure of the set F, every node v is assigned to one of the following:

(a) Some major monarch m at distance at most 2 from v (i.e., such that $v \in dom(m)$).
(b) Some minor monarch m' such that $m' \in minors(m)$ for some major monarch m, where $v \in \Gamma_2(m)$.
(c) Some new center $c \notin M$ such that $c \in Emp(Parent(m))$, where m is the major monarch such that $v \in Emp(m)$.
(d) $Parent(m)$, where m is the major monarch such that $v \in Emp(m)$.
(e) Some new center $c \notin M$ such that $c \in Emp(m)$, where m is the major monarch such that $v \in Emp(m)$.

It's not hard to verify that before the failure of the set F, all nodes were assigned to a center whose distance in G_W is at most 7. Consider a node v and let m be the center that served v before the failure of F. Let m' be the center that serves v after the failure of the set F.

First note that in Procedure ReAssign_by_F, every node may be reassigned at most once. Each such v_i is assigned to $\chi(f, m_{i+1})$ for $1 \leq i \leq j-1$. Note that $dist(v_i, m_i) \leq 5$ and $dist(m_i, \chi(f, m_{i+1})) \leq 4$, hence every v_i is assigned to a center at distance at most 9 from it, for $1 \leq i \leq j-1$. The last node v_j is assigned to $free(v)$. Note that $dist(v_j, m_j) \leq 5$ and $dist(m_j, free(v)) \leq 2$, so $dist(v_j, free(v)) \leq 7$. In conclusion, the distance is at most 9. □

By Lemmas 4.7 and 4.8 we have the following.

Corollary 4.9. *Algorithm Main yields an approximation ratio of 9 for the fault-tolerant capacitated K-center problem.*

5. Constant approximation for the α-FT-CCKC problem

We now present a constant approximation algorithm to the α-FT-CCKC problem. For the same reasons mentioned in Section 4, we consider only the case where the minimum node degree in G_W is at least α.

5.1. Relationship with α-FT-CKC

The following lemma shows that the α-FT-CCKC problem might requires creating more centers than the *CKC* problem. We say that a set of nodes $A \subseteq V$ is *k-independent* if every two nodes in A are at distance at least k apart. Let $K^*_{\text{CKC}}(W)$ be the minimal number of centers needed for a feasible solution of cost at most W for the CKC problem. Similarly, let $K^*_{\alpha\text{-FT-CCKC}}(W)$ be the minimal number of centers needed for a feasible solution of cost at most W for the α-FT-CCKC problem. Let $R^*_{\text{CKC}}(W)$ be a solution to the Capacitated K-Centers of cost at most W with minimal number of centers and let $R^*_{\alpha\text{-FT-CCKC}}(W)$ be a solution to the α-FT-CCKC of cost at most W with minimal number of centers.

Lemma 5.1. *For a 7-independent $A \subseteq V$, $K^*_{\alpha\text{-FT-CCKC}}(W) \geq K^*_{\text{CKC}}(W) + \alpha|A|$.*

Proof. Let $A = \{v_1, ..., v_k\}$ where $d(v_i, v_j) \geq 7$ for every $1 \leq i \neq j \leq k$. To prove the lemma, we show that it is possible to remove $\alpha|A|$ centers from $R^*_{\alpha\text{-FT-CCKC}}(W)$ and still have a feasible solution to the *CKC* problem of cost at most W. We first claim that $|\Gamma_1(v_j) \cap R^*_{\alpha\text{-FT-CCKC}}(W)| \geq \alpha + 1$ for every $1 \leq j \leq k$. To see this, assume otherwise, i.e., suppose there exists some $1 \leq j \leq k$ such that the optimal solution $R^*_{\alpha\text{-FT-CCKC}}(W)$ contains fewer than $\alpha + 1$ centers in $\Gamma_1(v_j)$. Then all centers in $\Gamma_1(v_j) \cap R^*_{\alpha\text{-FT-CCKC}}(W)$ might fail simultaneously, forcing v_j to be served by a backup center at distance greater than W, contradiction. In addition, the 3-neighborhoods $\Gamma_3(v_j)$ are disjoint for $1 \leq j \leq k$, due to the fact that A is 7-independent. For every $1 \leq j \leq k$, choose arbitrarily a set F_j of α nodes from $\Gamma_1(v_j) \cap R^*_{\alpha\text{-FT-CCKC}}(W)$. We now consider the failure of the set of centers F_j under the solution $R^*_{\alpha\text{-FT-CCKC}}(W)$. Let B_j be the set of servers that are assigned as backup centers to the nodes that belonged to $dom(F_j)$ before the failure. We now claim that $B_j \subseteq \Gamma_3(v_j)$. To see this, note that all clients of centers from $\Gamma_1(v_j)$ are at distance at most W from them, hence all clients of centers from $\Gamma_1(v_j)$ are in $\Gamma_2(v_j)$. The backup centers of all those clients are at distance at most W from the clients, therefore all these backup centers are in $\Gamma_3(v_j)$. As the sets $\Gamma_3(v_j)$ are disjoint, it follows that the sets B_j are also disjoint for $1 \leq j \leq k$. We now consider the solution R' obtained by taking $R^*_{\alpha\text{-FT-CCKC}}(W)$ and removing all nodes of F_j for every $1 \leq j \leq k$. As the sets B_j are disjoint and the centers of B_j can handle all nodes that were originally served by the centers of F_j, we conclude that R' is also a feasible solution to the Capacitated K-Centers problem. Hence $|R^*_{\alpha\text{-FT-CCKC}}(W)| \geq |R^*_{\text{CKC}}(W)| + \alpha|A|$. □

5.2. Multi-k-centers

Let us consider first a simpler variant of this problem, where a node may host several centers. Using Lemma 5.1 we can establish a 6-approximation ratio for this problem. It's not hard to see that the same bound as in Lemma 5.1 also holds in case multi-centers at each node are allowed. At the first stage, invoke the 5-approximation ratio algorithm of [14] for the multi-k-centers problem. Each node is now assigned to a center. Next, select backup centers and assign each node to α backup centers that will serve it in case its original center fails. In the second stage, choose a maximal 7-independent set of centers R, namely, every pair of nodes in R are at distance at least 7 from one another and for every node $v \in V$ there is a node in R at distance at most 6 from it. In each node $r \in R$, place α backup centers. Note that in case no failures occurred, the backup centers do not serve any node. Therefore, in case at most α centers fail, the backup centers can be used to replace the faulty centers. By Lemma 5.1, the number of centers that are selected in both these stages is at most $K^*_{\alpha\text{-FT-CCKC}}(W)$. Moreover, the distance from every node to its center is at most 5 and the distance to one of its backup centers is at most 6. Hence, we get a 6-approximation ratio. Therefore, the main difficulty in the version where we allow only one center at each node is to spread the centers in a way that will not harm the approximation too much.

5.3. The algorithm

The next lemma shows that if the α-FT-CCKC problem admits a feasible solution of cost at most W, namely, $K^*_{\alpha\text{-FT-CCKC}}(W) \leq K$, then for every node v there are sufficiently many nodes in $\Gamma_4(v)$ to allocate α backup centers (that will not serve any node as long as there are no failures) and sufficiently many additional centers to serve all nodes in $\Gamma_4(v)$. If this is not the case, namely, $\Gamma_4(v)$ is too small, then there is no feasible solution of cost at most W, and the algorithm has to proceed to the next possible weight W'. This will be needed in our algorithm when placing centers at the nodes.

Lemma 5.2. *If the α-FT-CCKC problem admits a feasible solution of cost at most W, then for every node v, the set $\Gamma_4(v)$ has sufficiently many nodes to allocate centers for all nodes in $\Gamma_4(v)$ and for α additional backup servers, i.e., $|\Gamma_4(v)| \geq \alpha + \lceil |\Gamma_4(v)|/L \rceil$.*

Proof. We first observe that if $N_3(v)$ is empty, then there must be enough available nodes in $\Gamma_2(v)$ to allocate as centers for all the nodes in $\Gamma_2(v)$ and for α backup servers. To see this, first note that if $\Gamma_2(v)$ contains fewer than α nodes than all centers located at $\Gamma_2(v)$ might fail simultaneously, forcing the nodes in $\Gamma_2(v)$ to be served by centers at distance more than W. Now if $\Gamma_2(v)$ contains at least α nodes but still fewer than $\alpha + \lceil |\Gamma_4(v)|/L \rceil$, then α centers might fail simultaneously at $\Gamma_2(v)$ and again some nodes at $\Gamma_2(v)$ will be served by centers at distance more than W.

So now assume $N_3(v)$ is not empty. As mentioned above, there must be at least $\alpha + 1$ servers in $\Gamma_1(v)$ in the optimal solution $R^*_{\alpha\text{-FT-CCKC}}(W)$. Consider some node u in $N_3(v)$. There must be at least $\alpha + 1$ servers in $\Gamma_1(u)$ in the optimal solution $R^*_{\alpha\text{-FT-CCKC}}(W)$. Note that $\Gamma_1(u) \subseteq N_2(v) \cup N_3(v) \cup N_4(v)$. Therefore, there must be at least α additional nodes in $N_2(v) \cup N_3(v) \cup N_4(v)$. Altogether, there are at least 2α nodes in $\Gamma_4(v)$. Since $L \geq 2$, the lemma follows. □

As in Section 4, we employ Procedures Select_Monarchs, Assign_Domains and ReAssign, using the same Procedure Assign_Domains as in [14] and modifying Procedures Select_ Monarchs and ReAssign.

As explained above, Procedure Select_Monarchs constructs a tree T of major monarchs. The idea behind assigning the monarchs in Procedure Select_Monarchs is as follows. In Section 4 we could settle for α backup centers, due to the assumption that all nodes can be reassigned after the failure of some centers. In contrast, in the current setting we need to spread many additional backup centers on the graph, in order to ensure that each node v has sufficiently many backup centers close to it. We select major monarchs at some distance from each other, and for each major monarch we allocate α of its neighbors as backup centers. In order to make sure that each major monarch can "afford" to allocate α backup centers and still have enough nearby nodes to allocate as centers to all nodes in its empire and to nodes passed to it, we first make sure that each major monarch m has all the nodes $\Gamma_4(m)$ in its empire. By Lemma 5.2 this guarantees that each monarch m has enough nodes to allocate centers to α backup centers and to handle all nodes in $\Gamma_4(m)$. To ensure that, we select the major monarchs so that they are at distance at least 10 from each other, and each monarch is at distance exactly 10 from its parent monarch. All nodes in $\Gamma_4(m)$ are assigned to m's empire and nodes at distance 5 from some major monarchs are assigned to the first selected monarch. For a major monarch m, define the deputy of m as some node that is at level-5 of m's parent monarch and its distance to m is 5, where a node v is on level-k of some monarch $\tilde{m}$ if v belongs to $Emp(\tilde{m})$ and v is at distance k from $\tilde{m}$. Note that in contrast to the setting in Section 4, here a node may be the deputy of more than one major monarch. In order to prove that a major monarch m has enough free nodes to allocate centers for all nodes passed to it from its children, we make sure each deputy will get at most $L-1$ nodes from the monarchs it serves as their deputy. For all other unassigned nodes, we allocate centers in the empire of some of these children.

Formally, the major monarchs are selected by an iterative process. Initially set Q to contain an arbitrary node v. While Q has unmarked nodes v such that $dist(v, M_1) \geq 10$ do the following. If $M_1 = \emptyset$ then remove a node v from Q (in this case Q contains only one node) else remove an unmarked node v such that $dist(v, M_1) = 10$ from Q. Make v a major monarch, add it to M_1 and mark it. Add all unmarked nodes in $\Gamma_5(v)$ to $Emp(v)$ and mark them. For each node w in $N_{10}(v)$ (distance 10 from v) such that there exists a node u in $Emp(v) \cap N_5(v) \cap N_5(w)$ do the following. If w is unmarked and $w \notin Q$ then set v to be w's parent in T, setting $Parent(w) = v$, set the deputy of w to be u, and add w to Q. In addition, for every node $m \in M_1$, choose α arbitrary nodes at distance 1 from m and make them backup centers.

Observe that for every $m \in M_1$, $\Gamma_4(m) \in Emp(m)$. In addition, note that after this process ends, there could be some nodes that are unassigned to any empire, namely, nodes that are at distance more than 5 from all major monarchs and thus were not selected to be in the empire of any major monarch. Observe that these nodes are at distance at most 9 from some major monarchs (as otherwise they would have been selected as major monarchs). The purpose of the second stage of Procedure Select_ Monarchs is to handle these unassigned nodes. In this stage we allocate minor monarchs and assign all unassigned nodes to those monarchs. This is again done by an iterative process as follows.

Algorithm Select_Monarchs(G_W) [for the conservative version]

1. set $M_1 \leftarrow \emptyset$, the set of major monarchs
2. set $M_2 \leftarrow \emptyset$, the set of minor monarchs
3. pick an arbitrary vertex $v \in V$ and set $Q = \{v\}$
4. $Parent(v) = nil$
5. while Q has unmarked nodes v such that $dist(v, M_1) \geq 10$ do
   - if $M_1 = \emptyset$ then remove a node v from Q else remove an unmarked node v such that $dist(v, M_1) = 10$ from Q
   - make v a major monarch, add it to M_1 and mark it
   - add all unmarked nodes in $\Gamma_5(v)$ to $Emp(v)$ and mark them
   - for each node u in $Emp(v) \cap N_5(v)$
     - for each node w in $N_5(u)$
       - if w is unmarked and $w \notin Q$ then set v to be w's parent in T, setting $Parent(w) = v$, set the deputy of w to be u, and add w to Q
6. for each $m \in M_1$, choose α arbitrary nodes at distance 1 from m and make them backup centers
7. $Q \leftarrow \emptyset$
8. for each $m \in M_1$
   - for each node u in $N_5(m) \cap Emp(m)$
     - for every unassigned neighbor m' of u such that $m' \notin Q$, set m to be m' parent, setting $Parent(m') = m$, add m' to Q, and set the deputy of m' to be u
9. while Q has unassigned nodes do
   - remove an unassigned node m from Q
   - make m a minor monarch, add it to M_2
   - assign all unassigned nodes from $\Gamma_5(m)$ to m's empire
   - for each node u in $N_5(m) \cap Emp(m)$
     - for every unassigned neighbor m' of u such that $m' \notin Q$, set m to be m' parent, setting $Parent(m') = m$, add m' to Q, and set the deputy of m' to be u

In each iteration, choose an unassigned node m' that is a neighbor of some node u such that $u \in N_5(m) \cap Emp(m)$ for some major or minor monarch m. Make m' a minor monarch, place it in M_2 and assign m' all unassigned nodes at distance 5 from it. Set the parent of m' to be $Parent(m') = m$ and the deputy of m' to be u. In the end of this process, all nodes are assigned to the empire of some monarch and the distance from a minor monarch to its parent is 6. Let M_2 be the set of minor monarchs.

Procedure Assign_Domains is again similar to the one presented in [14]. The procedure assigns as many nodes as possible to all chosen major and minor monarchs, where once again, each monarch can serve all nodes at distance at most 2 from it. The domain $dom(m)$ of a monarch m is the set of all nodes assigned to it by this procedure.

Algorithm Assign_Domains(G_W) [for the conservative version]

1. let M be the set of monarchs in G_W
2. let $E' = \{(m, v) | m \in M, v \in V,$ and $dist(v, m, G_W) \leq 2\}$
3. construct a bipartite graph $\tilde{G} = (M, V, E')$
4. add vertices s and t, add edges from s to all $m \in M$ and from all $v \in V$ to t
5. set capacities $u(s, m) = L$, $u(m, v) = 1$ and $u(v, t) = 1$ for $m \in M$ and $v \in V$
6. compute a maximum integral flow in $\tilde{G}$
7. for each monarch m,
   set $dom(m) = \{v \mid v$ receives 1 unit flow from $m\}$

Procedure ReAssign takes care of all nodes that are not served by any center. The main difference with respect to Procedure ReAssign in Section 4 is that here, a node may be the deputy of more than one monarch. We need to make sure each deputy receives at most $L - 1$ nodes in total from all the monarchs it serves as deputy. We do that by allocating centers in the empires of some of these children. Formally, let $unassigned(m)$ be the set of all nodes in its empire that are unassigned, namely the nodes in its empire that do not belong to the domain of any monarch. Let T be the tree of monarchs. The process maintains a copy T' of T consisting of all the monarchs that have not been handled yet. In each step of the algorithm, pick a leaf m from the tree T' for processing and remove it from T'. Now for each node u at level-5 of m, let $kL + \epsilon$ be the number of nodes passed to m by its children monarchs in T such that u serves as their deputy. If $k > 0$ then allocate k new centers in the empires of the monarchs that u serves as their deputy. Assign kL from $passed(u)$ to those new centers. Pass the ϵ remaining nodes to m. The next step takes care of all unassigned nodes in m's empire and

the nodes passed to m by its children in the tree T. Let $k'L+\epsilon$ be the number of nodes in m's empire that are unassigned plus the number of nodes passed to m. Allocate k' new centers in the empire of monarch m. Allocate the ϵ remaining nodes at m possibly displacing ϵ other nodes from m's original clients. Add the displaced nodes to the list of Passed nodes of the deputy of m unless m is the root and then allocate a new center at m's empire and assign the unassigned nodes to it.

Algorithm ReAssign(G_W) [for the conservative version]

1. let $M = M_1 \cup M_2$ be the set of monarchs in G_W
2. for each monarch $m \in M$, set $unassigned(m) = Emp(m) \setminus dom(M)$
3. let T be the tree of monarchs
4. for each node $u \in V$, set $passed(u) \leftarrow \emptyset$
5. while T is not empty do:
   - remove a leaf m from T
   - for each node u at level-5 of m do:
     - let $|passed(u)| = kL+\epsilon$
     - if $k > 0$ then allocate k new centers in the empires of the monarchs that u serve as their deputy. Assign kL from $passed(u)$ to those new centers
     - pass the ϵ remaining nodes to m
   - let $|passed(m) \cup unassigned(m)| = kL+\epsilon$
   - allocate k new centers in the empire of monarch m
   - allocate the ϵ remaining nodes at m possibly displacing ϵ other nodes from m's original clients
   - add the displaced nodes to the list of Passed nodes of the deputy of m unless m is the root and then allocate a new center at m's empire and assign the unassigned nodes to it

5.4. Analysis

Towards proving that the number of centers selected by our algorithm is at most the number required in the optimal solution $R^*_{\alpha\text{-FT-CCKC}}(W)$, we first give the following lemma. As in Section 4, K_L denotes the number of light monarchs and n_L the number of vertices belonging to the domains of light monarchs, and n is the total number of vertices. K_L, n_L and n are defined as in Section 4. Note that we do not consider the backup centers as monarchs, and that no clients are assigned to the backup centers before a failure event happens. Denote by $dom^*_{\text{CKC}}(\theta)$ the set of nodes that are served by some center θ in the optimal solution $R^*_{\text{CKC}}(W)$ for the *CKC* problem.

Lemma 5.3. *The number of centers required in the optimal solution $R^*_{\text{CKC}}(W)$ is at least $K_L + \lceil (n-n_L)/L \rceil$.*

Proof. Notice that each monarch has a unique center that covers it in the optimal solution $R^*_{\text{CKC}}(W)$. Consider a light monarch m. let θ be the center that covers m in the optimal solution $R^*_{\text{CKC}}(W)$. We claim that, $dom^*_{\text{CKC}}(\theta) \subseteq dom(m)$. Assume otherwise, let u be some node that θ covers in the optimal solution $R^*_{\text{CKC}}(W)$ and m does not cover it. Notice that as the monarchs are at distance at least 6 from each other, then $dist(m', u) > 2$ for any monarch $m' \neq m$ and therefore u cannot be assigned to any $m' \neq m$. We get that u does not belong to any domain. Since m is a light monarch and its distance to u is less than 2, then m can be assigned as a center to u, resulting a higher flow, a contradiction. □

By Lemmas 5.1 and 5.3, we have the following.

Lemma 5.4. $K^*_{\alpha\text{-FT-CCKC}}(W) \geq K_L + \lceil (n-n_L)/L \rceil + \alpha|M_1| = K_W$.

The following two lemmas show that each time the algorithm has to allocate new centers, there are sufficiently many free nodes to do so.

Lemma 5.5. *Each monarch m has sufficiently many available nodes in its vicinity to allocate centers for $passed(m)$ and $unassigned(m)$ nodes.*

Proof. We need to consider two cases. The first is when m is a major monarch, i.e., $m \in M_1$. Notice that in that case, $Emp(m)$ contain all nodes from $\Gamma_4(m)$ and possibly some other nodes from $N_5(m)$. By Lemma 5.2 there are sufficiently many available nodes in $\Gamma_4(m)$ to allocate centers for all nodes in $\Gamma_4(m)$ and also for α backup servers (note that $dom(m) \subseteq \Gamma_2(m) \subseteq \Gamma_4(m)$). Each node in $N_5(m)$ passes to m at most $L-1$ nodes from the monarchs it serves as their deputy. In addition, the nodes in $N_5(m)$ do not host a center, so they are available to be allocated as centers. The second case is when

m is a minor monarch, i.e., $m \in M_2$. In that case, the algorithm does not place backup servers on m's empire, so clearly there are enough available nodes to allocate as centers for all nodes. □

Lemma 5.6. *Let u be the deputy of a set of monarchs S and assume $|passed(u)| = kL + \epsilon$ for some integer $k \geq 0$. Then there are at least k available nodes to allocate centers in the empires of S.*

Proof. By Lemma 5.5, each monarch $t \in S$ has sufficiently many available nodes to allocate centers for the clients in $passed(t)$ and $unassigned(t)$ nodes. So if some $t \in S$ passes nodes to u, then there is left an available node at $Emp(t)$ to allocate as a new center. So there are at least k available nodes to allocate as centers in the empires of S. □

Finally, the following lemma establishes the desired stretch bound.

Lemma 5.7. *Under all possible subsets of up to α failed service centers, each vertex v is assigned to a center w s.t. $dist(v, w) \leq 17$ in G_W.*

Proof. Consider a client v and let m be its monarch, namely, $v \in Emp(m)$. If v is not passed in Procedure ReAssign, then it is still assigned to m and note that $dist(v, m) \leq 2$. Assume v was passed to another center in Procedure ReAssign. We consider several cases.

The first case is when $m \in M_1$ and v is covered by some node in $Emp(Parent(m))$. Notice that $dist(v, m) \leq 2$, $dist(m, Parent(m)) \leq 10$ and $dist(Parent(m), z) \leq 5$ for every $z \in Emp(Parent(m))$. Therefore, the distance from v to the center it is assigned to satisfies $dist(v, ctr(v)) \leq 17$.

The second case is when $m \in M_1$ and v is covered by some node in the empire of one of its brother monarchs m' where $deputy(m) = deputy(m')$. Again, $dist(v, m) \leq 2$, $dist(m, m') \leq 10$ and $dist(m', z) \leq 5$ for every $z \in Emp(m')$, so $dist(v, ctr(v)) \leq 17$.

The third case is when $m \in M_2$ and v is covered by some node in $Emp(Parent(m))$. Again, $dist(v, m) \leq 2$ and $dist(m, Parent(m)) \leq 6$, so $dist(v, ctr(v)) \leq 13$.

The last case is when $m \in M_2$ and v is covered by some node in the empire of one of its brothers monarchs. Again, $dist(v, m) \leq 2$. So $dist(v, ctr(v)) \leq 13$.

Also notice that each node has α backup servers at distance at most 10 from it. So the maximum distance is at most 17. □

5.5. Large capacities

We now consider the special case where $\alpha < L$.

Lemma 5.8. *For every node v, the set $\Gamma_1(v)$ has sufficiently many available nodes to allocate centers for all nodes in $\Gamma_1(v)$ and for α backup servers.*

Proof. There must be at least $\alpha + 1$ servers in $\Gamma_1(v)$ in the optimal solution $R^*_{\alpha\text{-FT-CCKC}}(W)$. Consider some α backup centers from $N_1(v)$. The node v can cover itself and also the α backup centers from $N_1(v)$. All other uncovered nodes do not contain a center placed on them and are free to be allocated as centers. □

We use the same method as in the general case, with some changes. The set of monarchs M_1 is chosen to be 7-independent instead of 10-independent. The empire of each monarch $m \in M_1$ is set to be all nodes unassigned so far from $\Gamma_3(v)$, instead of $\Gamma_5(v)$. The monarchs M_2 are chosen in the same manner as before, but handle all nodes unassigned so far at distance 3 from them instead of 5. By the same reasoning as before, we bound the approximation ratio by 13.

References

[1] H.-C. An, A. Bhaskara, C. Chekuri, S. Gupta, V. Madan, O. Svensson, Centrality of trees for capacitated k-center, in: Proc. 17th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO), 2014, pp. 52–63.
[2] J. Bar-Ilan, G. Kortsarz, D. Peleg, How to allocate network centers, J. Algorithms 15 (1993) 385–415.
[3] J. Bar-Ilan, G. Kortsarz, D. Peleg, Generalized submodular cover problems and applications, Theoret. Comput. Sci. 250 (2001) 179–200.
[4] M. Cygan, M. Hajiaghayi, S. Khuller, LP rounding for k-centers with non-uniform hard capacities, in: Proc. 53rd IEEE Symp. on Foundations of Computer Science, (FOCS), 2012, pp. 273–282.
[5] M. Dyer, A.M. Frieze, A simple heuristic for the p-center problem, Oper. Res. Lett. 3 (1985) 285–288.
[6] J. Edmonds, D.R. Fulkerson, Bottleneck extrema, J. Combin. Theory 8 (1970) 299–306.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1978.
[8] T. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoret. Comput. Sci. 38 (1985) 293–306.
[9] D.S. Hochbaum, D.B. Shmoys, Powers of graphs: a powerful approximation algorithm technique for bottleneck problems, in: Proc. 16th ACM Symp. on Theory of Computing, 1984, pp. 324–333.
[10] D.S. Hochbaum, D.B. Shmoys, A best possible heuristic for the k-center problem, Math. Oper. Res. 10 (1985) 180–184.

[11] D.S. Hochbaum, D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, J. ACM 33 (3) (1986) 533–550.
[12] W.L. Hsu, G.L. Nemhauser, Easy and hard bottleneck location problems, Discrete Appl. Math. 1 (1979) 209–216.
[13] S. Khuller, R. Pless, Y. Sussmann, Fault tolerant k-center problems, Theoret. Comput. Sci. 242 (2000) 237–245.
[14] S. Khuller, Y. Sussmann, The capacitated K-center problem, SIAM J. Discrete Math. 13 (2000) 403–418.
[15] J. Plesnik, A heuristic for the p-center problem in graphs, Discrete Appl. Math. 17 (1987) 263–268.