$O(1)$-Approximations for Maximum Movement Problems

Piotr Berman[1], Erik D. Demaine[2], and Morteza Zadimoghaddam[2]

[1] Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802, USA
[2] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Abstract. We develop constant-factor approximation algorithms for minimizing the maximum movement made by pebbles on a graph to reach a configuration in which the pebbles form a connected subgraph (connectivity), or interconnect a constant number of stationary nodes (Steiner tree). These problems model the minimization of the total time required to reconfigure a robot swarm to achieve a proximity (e.g., radio) network with these connectivity properties. Our approximation factors are tight up to constant factors, as none of these problems admit a $(2-\varepsilon)$-approximation assuming $\mathrm{P} \neq \mathrm{NP}$.

1 Introduction

A central problem in swarm robotics is to reconfigure the robots into an arrangement with a desired property. For example, in the *connectivity* goal, the proximity of the robots should form a connected graph. Two motivations for this goal are forming a connected data network with short-range radios, and forming a connected physical network for transporting materials. In the first situation, the robots initially communicate via another channel, e.g., via slow and/or power-intensive long-distance communication (such as satellite or the same radios with power turned up high), or via two traversals by aircraft to locate robots and disseminate locations. Another connectivity goal is *Steiner connectivity*, where a subset of the robots should form a connected network that contains k stationary nodes (such as buildings or sensors) which need to be interconnected. In both of these problems, we suppose that we know the initial locations of robots, and that we have a map of the environment the robots can traverse and defining proximity among the robots. Our goal is to move the robots as quickly as possible into a configuration with the desired property.

These problems fit into the broad family of *movement problems*, introduced in [5], and further explored in [6,8]. In general, we have a graph G with pebbles on some of the vertices. The goal is to move some of the pebbles along edges to obtain a desired property P, while minimizing either the maximum movement or the total movement of the pebbles, motivated by minimizing either execution time or energy usage. Specific problems considered in [5] include the *connectivity movement problem* mentioned above, where property P is that the vertices with at least one pebble induce a connected graph, and the *s-t path movement problem*, where property P is that the vertices with at least one pebble induce a graph in which two given vertices are in a common connected component (the special case of the Steiner connectivity movement problem with two terminals).

Several approximation algorithms and inapproximability results for these movement problems were presented in [5]. Of primary relevance to this paper, the connectivity and s-t path movement problems with the maximum movement objective function have an $O(\sqrt{n})$-approximation algorithm.[3] Furthermore, both of these problems are $(2-\varepsilon)$-inapproximable assuming $\mathrm{P} \neq \mathrm{NP}$. On the other hand, with the total movement object function, the connectivity movement problem is $\Omega(n^{1-\varepsilon})$-inapproximable, and there is an $\tilde{O}(n)$-approximation algorithm. This negative result motivates our focus here on the maximum movement objective.

In FOCS 2008, Friggstad and Salavatipour [8] considered a *facility location movement problem*: moving pebbles of two types, facilities and clients, to achieve the property that each client is at a vertex that contains some facility. This problem was first introduced in [5], which presented a simple 2-approximation algorithm for this problem with the maximum movement objective function. But the problem with the total movement objective remained an open problem until Friggstad and Salavatipour developed an LP-based constant-factor approximation algorithm.

Demaine et. al. characterize the tractable and intractable movements problems in a general setting in [6]. They consider the class of edge-deletion minimal graphs that have the desired property P, and can be the final destination of the pebbles, e.g. for connectivity this class is the set of subtrees of graph G with at most m nodes where m is the number of pebbles. They prove that if for property P the treewidth is bounded by a constant, the movement problem of this property is Fixed Parameter Tractable. They consider the number of pebbles as the parameter of their algorithms. So their results are applicable only for small number of pebbles, e.g. $m = O(\log(n))$.

As some applications of these movement problems in Wireless Networks, we can refer to the works of Basagni et. al. who consider movements of some mobile sinks in order to maximize the network lifetime [1,2,3]. There are several fixed (non-mobile) sensor nodes with limited power. The mobile nodes should move between some fixed sites to gather data from the sensor nodes. We have to find the best schedule for mobile sinks' movements to maximize the network lifetime, i.e. lifetime is equal to the time duration in which all nodes have enough energy supply to handle all their operations including sensing, transmitting, and etc.

Our Results. Connectivity is the quintessential movement problem. For the total movement objective function, the best possible approximation ratio is known to be roughly linear [5] (assuming $\mathrm{P} \neq \mathrm{NP}$). But for the maximum movement objective function, which models the parallel execution time or makespan required for the motion, there is a huge gap in the best possible approximation ratio, between $2-\varepsilon$ and $O(\sqrt{n})$. In this paper, we close this gap up to constant factors, by obtaining the first constant-factor approximation algorithm for the connectivity movement problem.

An ingredient in this result is a constant-factor approximation algorithm for the s-t path movement problem. This result is also a breakthrough, as again the best possible approximation ratio was previously only known to be between $2-\varepsilon$ and $O(\sqrt{n})$. We use our approximation algorithm for s-t path problem as a black box in our solution to the connectivity problem.

Finally we introduce the *Steiner connectivity movement problem*, which is a natural generalization of the s-t path movement problem. Here we are given a set T of terminal

[3] In fact the approximation ratio is $O(\sqrt{m/\mathrm{OPT}})$ where m is the number of pebbles and OPT is the maximum movement of the optimum solution. This ratio can be as large as $\Theta(\sqrt{n})$, when $m = \Theta(n)$ and $\mathrm{OPT} = \Theta(1)$.

vertices, and the goal is to move some of our pebbles to interconnect all terminal vertices. More precisely, property P is that the vertices with at least one pebble induce a graph that places all terminals in the same connected component. For $|T| = O(\log n)$, we present an $O(|T|)$-approximation algorithm for the Steiner connectivity movement problem with the maximum movement objective, again using our approximation algorithm for s-t path. Note that we cannot hope to approximate the Steiner connectivity movement problem for arbitrary $|T|$, because even deciding whether there is a feasible solution with the available pebbles is the NP-hard node-weighted Steiner tree problem. Unfortunately we include this result only in the complete version because of space limits.

Techniques. Our algorithms introduce several new techniques for approximating movement problems with the maximum movement objective. In general, these problems would be easy if we allowed approximating the number of pebbles (via resource augmentation) in addition to the cost. But robots cannot (yet) replicate themselves, so resource augmentation is not very useful. We develop powerful tools to resolve multiple desires for the location of a single pebble/robot.

In the s-t path movement problem (Section 2), we define a concept of a *locally consistent paths* such that (a) a correct solution is also locally consistent, (b) the minimum length locally consistent solution can be found in polynomial time, and (c) with only limited additional pebble movement, it can be converted to a consistent solution. For the sake of simplicity, we describe an algorithm with an extremely large polynomial time bound that achieves a 7-approximation; in the full paper, we will describe the available trade-off between the running time and the approximation of the maximum movement.

In the connectivity movement problem (Section 3), we present a three stage algorithm. In all stages, we maintain a set of pebbles S (initially empty), and try to move some pebbles and insert them into S. The new locations of pebbles in S form a connected subgraph. In the first stage, we define dense vertices which are basically the vertices with a large enough number of pebbles around them. Once we find a dense vertex, we can use the s-t path algorithm, and the pebbles around the dense vertex to insert a subset of pebbles into set S. We do this process iteratively in the first stage until there exists no dense vertex.

Then we analyze the structure of the remaining pebbles in the optimal solution. The final locations of all pebbles in the optimal solution is a connected subgraph and has a spanning tree T. In Figure 1 (page 7), you can see an instance of our problem, and its optimal solution on the left. The spanning tree T is shown with bold edges on the right. After the first stage, some pebbles are inserted into S, we remove these pebbles from tree T. The remaining subgraph is a forest F with some interesting properties. We prove that there can not be a vertex in this forest with three long paths attached to it, call such a vertex "tripod vertex", e.g. vertex v in Figure 1. We prove that after the first stage there exists no tripod vertex in forest F. We then prove that every subtree in F either has low diameter or is a long path with some low-diameter branchlets attached to it, call it a thick path tree[4]. If the subtree has low diameter, we prove that all its pebbles are close to set S, so we can insert them into S by moving them directly toward this set. If it is a thick path tree, we enumerate to find the head and tail of its longest path, then we use s-t path algorithm to connect its head and tail using its own pebbles. We also prove that every other pebble in this subtree is close to the path we find (we need the second stage for proving this claim), and we can move all these pebbles to connect them to set S with $O(M)$ movement where M is the maximum movement of an optimum solution.

[4] We call it Caterpillar Shape tree as well.

We note that one can enumerate on all possible values of M which are $1, 2, \cdots, n$, or use binary search to find it. So we can assume that our algorithm known the value of M.

The important problem is that these subtrees are not easily distinguishable. Since pebbles from different subtrees can be close to each other in their starting configuration, we can not find out which pebbles are in which subtree. We know that two adjacent pebbles in the optimum solution have distance at most $M + 1 + M = 2M + 1$ from each other in their starting configuration. We make graph H with the remaining pebbles (outside set S) as its vertices. We put an edge between two pebbles if their distance is at most $2M + 1$ from each other. This way we can be sure that all pebbles in a subtree of F are in the same connected component in H. But there might be pebbles from several subtrees of F in the same connected component in H.

We define some relaxed versions of dense vertices, and try to insert more pebbles into set S by finding these vertices in the second stage. In the third stage, we find all remaining pebbles close to set S, and insert them to set S by moving them toward S. The second stage helps us prove that there can not be pebbles from more than two thick path trees in a connected component of H. We can distinguish two thick path trees in a common connected component of H with some other techniques in polynomial time. But there might be pebbles from several low diameter subtrees in this connected component as well. We prove that all pebbles in low diameter subtrees are close to set S, and in the third stage we take care of all of them. So after the third stage, we might have some connected components in H containing up to 2 thick path trees. If the connected component has no thick path tree, we can show that all its pebbles are close to set S, so we just move them toward set S. If it has one thick path tree, we can enumerate to find the head and tail of the longest paths of this thick path tree, and run our s-t path algorithm to take care of its pebbles. If it has two thick path trees, we prove that all pebbles from the two thick path trees that are close to each other are around the tails of the longest paths of these two thick path trees. So we enumerate to find the tail of one of the thick path trees, and remove all pebbles in the vicinity of the tail vertex. This way we can distinguish between the two thick path trees, and handle each of them by the s-t path algorithm.

2 s-t Path Movement Problem

In the s-t path movement problem, we are given a graph G with two vertices s and t, and a set of pebbles with positions on the vertices of our graph. We want to move some pebbles to construct a path between s and t with at least one pebble on each of its vertices. Our objective is to minimize the maximum movement. The following theorem presents a constant-factor approximation algorithm for this problem.

Theorem 1. *There is a polynomial-time algorithm that finds solution to the s-t the path movement problem such that if there exists a solution that forms a path of length ℓ and moves each pebble along at most M edges, then the algorithm finds a path of length at most ℓ and which moves each pebble along at most $7M-4$ edges.*

Note that we do not move pebbles more than a constant factor of the maximum movement in the optimal solution, and at the same time we use no more pebbles than the optimal solution to construct a path between s and t. So our algorithm can be seen as a bicriteria approximation algorithm: it is optimal in terms of the number of pebbles used in the path, and it is a constant factor approximation with respect to the maximum

movement. The fact that we are optimal in terms of the number of pebbles used in the path helps us later to find approximation algorithms for the Steiner connectivity movement problem.

Our algorithm for the s-t path movement problem has two main parts which can be described as follows (with a bit of oversimplification):

1. Find a minimum length locally consistent path $\mathbb{P}$. A pebble can be moved along at most $3M-2$ edges to a node of $\mathbb{P}$, but we allow to move a pebble to multiple nodes on $\mathbb{P}$, provided that they are sufficiently far apart (about $14M$ edges).
2. Convert $\mathbb{P}$ to a consistent path $\mathbb{Q}$ as follows: for each pebble moved to multiple nodes of $\mathbb{P}$ select one of them, which creates gaps in the path; then fill the gaps by moving some pebbles along additional $(7M-4)-(3M-2) = 4M-2$ edges. The length of the path cannot increase.

Because the optimum solution, say of length ℓ, is locally consistent, the length of the locally consistent path that we will find cannot be larger than ℓ, and as we shall see, local consistency is easier to assure than actual consistency. In the same time, under our assumption each inconsistency offers an opportunity of making a shortcut: if a pebble can be moved by distance at most $3M-2$ to two locations that are $14M$ apart on the path, a section of the path with length $14M$ can be replaced with a shorter path with length $6M-4$, provided that we move pebbles from the longer section to the shortcut. This presents a challenge, of course, how to do it in a consistent manner.

2.1 Straighter Paths

We will use the following notation: $d(u, v)$ is the distance from u to v (the length of a shortest path), disk $D(u, R)$ is the set $\{v \in V : \ d(u, v) \leq R\}$, and a disk set $P(u, R)$ is the set of pebbles with the initial location within $D(u, R)$.

A valid solution has a path $\mathbb{P} = (s = u_0, u_1, \ldots, u_\ell = t)$ so that for $i = 0, \ldots, \ell$ there exists a pebble p_i such that $p_i \in P(u_i, M)$, moreover, $p_i \neq p_j$ for $0 \leq i < j \leq \ell$.

Given such a solution $\mathbb{P}$, we can find another, $\mathbb{Q}$, which we will call *the straighter path.*

We define *milestones* of $\mathbb{P}$ as a subsequence $v_1, \ldots, v_f$ of $\mathbb{P}$, and in turn they define $\mathbb{Q}$, a straighter version of $\mathbb{P}$, that connects s, t and the milestones using the shortest paths: from s to v_1, from v_1 to v_2, etc., and finally from v_f to t. A milestone v_i is responsible for the section of $\mathbb{Q}$ that is contained in $D(v_i, M-1)$ and which consists of r_i nodes, where $r_i = 2M-1$ for $i < f$, while $r_f \leq 2M-1$ is the distance from t to $D(v_{f-1}, M-1)$.

The initial milestone v_1 is u_j such that $j = \max\{k : \ d(s, u_k) \leq M-1\}$. Assume that milestone v_i is defined. If $d(v_i, t) \leq M-1$ then v_i is the final milestone. If $M \leq d(v_i, t) \leq 2M-1$ then $v_{i+1} = t$ is the final milestone. If $d(v_i, t) \geq 2M-1$ then $v_{i+1} = u_j$ where $j = \max\{k : \ d(v_i, u_k \leq 2M-1\}$.

We modify the pebble movement as follows: consider a node u' of $\mathbb{Q}$ that is a steps before milestone v (or after), where $a < M$. Then we have node u of $\mathbb{P}$ that is a steps before v (or after) on $\mathbb{P}$ and the pebble p that was moved to u (along at most M edges). The modified movement of p traverses a path to u, then to v and finally to u', hence it uses at most $m + 2a \leq 3M-2$ edges.

Now suppose that we do not know $\mathbb{P}$ or $\mathbb{Q}$ but only the milestones $v_1, \ldots, v_f$. We can find the $\mathbb{Q}$ by connecting s, t and the milestones with shortest paths (between the milestones).

A solution of that form exists if and only if there exist pairwise disjoint sets of pebbles $S_1, \ldots, S_f$ such that $S_i \subset P(v_i, 2M{-}1)$ and $|S_i| = r_i$. This justifies the following definition:
A *consistent solution* is a sequence of nodes $v_1, \ldots, v_f$ and a sequence of sets $S_1, \ldots, S_f$ such that

1. $d(v_1, s) = M{-}1$ and $d(v_i, v_{i+1}) = 2M{-}1$ for $i = 1, \ldots, f-2$;
2. either $d(v_{f-1}, v_f) = 2M{-}1$ and $d(v_f, t) < M$ or $v_f = t$ and $d(v_{f-1}, t) < 2M$;
3. $r_i = 2M{-}1$ for $i = 1, \ldots, f-1$ and $r_f = d(v_f, v_{f-1}) + d(v_f, t) - M + 1$;
4. $S_i \subset P(v_i, 2M{-}1)$ and $|S_i| = r_i$;
5. $S_i \cap S_j = \varnothing$ for $i \neq j$.

A locally consistent solution satisfies a weaker version of condition 5: $S_i \cap S_j = \varnothing$ for $i < j < i + 7$. The length (or the number of pebbles that are used) is $\sum_{i=1}^{f} r_i$.

Lemma 1. *There exists a polynomial time algorithm that find a locally consistent solution with the minimum length.*

Lemma 2. *In polynomial time we can transform a locally consistent solution $\mathbb{Q}$ of length ℓ to a solution that uses at most ℓ pebbles that are moved along at most $7M{-}4$ edges.*

To conclude the proof of Theorem 1, observe that we can perform the algorithm described in Lemma 1 for different values of M, to find the smallest value for which it concludes successfully, and then we finish using the algorithm described in Lemma 2.

3 Connectivity Movement Problem

In the *connectivity movement problem*, we want to move pebbles so that their final positions induce a connected subgraph of graph G. The goal is to minimize the maximum movement of pebbles. Without loss of generality we can assume that a vertex r is given, and the induced subgraph of the final positions of pebbles should contain r, i.e., there should be a path from the root r to all pebbles in their subgraph. The following theorem is the main result of this section:

Theorem 2. *Given a polynomial-time λ-approximation algorithm for the s-t path movement problem, we can construct a polynomial-time $(8\lambda + 80)$-approximation algorithm for the connectivity movement problem.*

Definition 1. *Suppose there are m pebbles: $p_1, p_2, \ldots, p_m$. Let u_i be the starting position of pebble p_i, and v_i be its final destination in the optimum solution. Vertices $v_1, v_2, \ldots, v_m$ form a connected subgraph in G. This connected subgraph has a spanning tree. Let T be one of its spanning trees. So tree T has at most m vertices because there is at least one pebble on each vertex in T in the optimal solution. In Figure 1, you can see a sample graph before the movements on the left, and with maximum movement one edge, all pebbles form a connected subgraph. Tree T is shown on right by bold edges.*

Because of lack of space, we present main ideas in this paper, and include detailed description and proofs in the complete version.

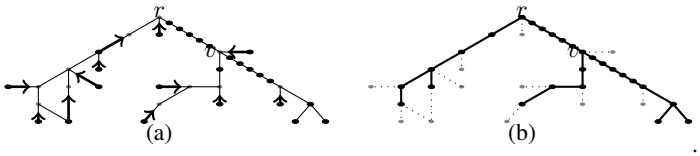

Fig. 1. Each black node contains a pebble, and gray nodes are vertices without pebbles. The arrows show the movements of pebbles in the optimum solution. The maximum movement is one.

3.1 A Constant factor Approximation for Connectivity Movement Problem

During our algorithm, the set S consists of all pebbles that are connected to root r (via other pebbles) up to now, i.e., this set might be empty at the beginning. We try to add pebbles into S by moving them. Our algorithm basically has three main phases. In Phase 1, we define Operations 1 and 2. We perform these two operations iteratively as long as we can perform them. During these two operations, whenever we move a pebble p, and add it to set S, we make sure that there exists a pebble $p' \in S$ such that its current position has distance at most λM from the starting position of pebble p. Note that pebble p' might or might not be the same as pebble p (in Operation 1 they are the same). When we cannot perform any of these two operations anymore, we can prove some interesting properties about the structure of the remaining pebbles (pebbles outside S).

Then in the second phase, we introduce new Operations 3 and 4, and iteratively perform them. We should note that Operations 3 and 4 are very similar to Operations 1 and 2, but existence of these two similar phases is essential in our algorithm. In the second phase when we move a pebble p, and add it to S, we make sure that there exists a pebble $p' \in S$ that its current position has distance at most xM from the starting position of pebble p where x is a parameter we define later in the second phase.

In the last phase, we present Operation 5, and perform it once. Then we show how to decompose the remaining pebbles into some groups, and take care of each group. In this phase we connect all remaining pebbles to set S, and therefore our connecting task is complete. The maximum movement is a constant times the optimum maximum movement M.

In Operation 2 of the first phase, we define dense vertices, and while there exists a dense vertex, we find a path from it to root r. Using this path we add some pebbles into set S. We iteratively perform Operations 1 and 2 if one of them is possible to do. If both of them are possible to do, Operation 1 has priority. Before performing the following operations, we move all pebbles with distance at most M from root r toward r. We also add them to set S. This way we make sure that S is not empty, and the following operations are feasible.

Operation 1. While there is a pebble $p \notin S$ whose distance to current position of some pebble $p' \in S$ is at most $\lambda M + 1$, we move pebble p towards pebble p' to make them neighbors. We can do this by moving p at most λM edges. Now we can add p to S.

Operation 2. Here we explain the main ideas. Vertex v is called dense, if there are at least $(2\lambda + 2)M$ pebbles outside set S with distance at most λM from v. Let A be the set of the pebbles outside S with distance at most λM from v. If there exists a dense vertex v (with $|A| \geq (2\lambda + 2)M$), we do the following. We gather all pebbles of A on vertex v. This can be done by moving these pebbles along at most λM edges. If the

distance of v from at least one pebble in S is at most $(2\lambda+2)M$, we can use the pebbles in A to connect vertex v to set S. This way we add pebbles of A to set S by moving them along at most $\lambda M + (2\lambda + 2)M = (3\lambda + 2)M$.

Otherwise (if v has distance greater than $(2\lambda+2)M$ from all pebbles in S), we know that a subset of pebbles outside S and A form a path from a pebble in A to a pebble in S in the optimal solution with maximum movement M. This path should start from a vertex with distance at most $\lambda M + M$ from v to a vertex with distance at most $M + \lambda M$ from set S. So we use our path movement algorithm to build this path. Then we use the pebbles of A to connect v to the first part of this path, and also connect the last part of this path to set S. So we can connect and join all these pebbles to S with maximum movement at most $\lambda M + 2(\lambda + 1)M = (3\lambda + 2)M$.

We keep doing the above operations until there is no dense vertex, and no pebble close to set of pebbles S. Following we show some interesting facts about the structure of the optimal solution after deleting pebbles of S when the first phase is finished, and we cannot perform Operations 1 and 2.

We keep vertices in tree T (defined in Definition 1) that have at least one pebble outside set S and remove the rest of them. The remaining graph is a forest, call it F. Suppose this forest has k connected components $T_1, T_2, \ldots, T_k$. Each of these k connected components is a subtree of T.

Lemma 3. *For each tree T_i, at least one of the following two conditions holds: (1) T_i has diameter less than $2(\lambda - 1)M$. (2) Every vertex in T_i has distance less than $4M$ from the longest path in T.*

Now that we cannot perform Operations 1 and 2 anymore, we start the second phase. In this phase, we define Operations 3 and 4, and we perform them iteratively. Like before, we stop when we cannot perform Operations 3 and 4 anymore. Define x to be $2\lambda + 22$ in the rest of the algorithm.

Operation 3. Consider a vertex v with distance $dis(v, S)$ from set S, i.e., $dis(v, S)$ is the minimum distance of v from pebbles in S. For $0 \leq i \leq xM$, let $N(v, i)$ be the number of pebbles outside S that has distance at most i from v. If $N(v, i)$ is at least $dis(v, S) - (xM - i)$ for some $0 \leq i \leq xM$, and $dis(v, S)$ is at most $(2x + 2)M$, we can do the following. We can gather all pebbles with distance at most i from v, on vertex v, and move them along the shortest path from v to set S to form a path attaching to the current set S. This way we can add some pebbles to S. We cannot necessarily fill the shortest path from v to S completely, but because $N(v, i)$ is at least $dis(v, S) - (xM - i)$, we lack at most $xM - i$ pebbles. Because we moved each pebble at most i in the gathering part, we can say that the starting position of each moved pebble has distance at most $i + (xM - i) = xM$ from the current position of a pebble in S (consider the last pebble in the path we just made and attached to set S). The maximum movement in this operation is at most $i + dis(v, S)$ which is at most $xM + (2x + 2)M = (3x + 2)M$. The interesting property of this operation is that we can use it for different values of $0 \leq i \leq xM$.

Operation 4. Vertex v is called dense with diameter xM, if there are at least $(2x + 2)M$ pebbles outside set S with distance at most xM from v. Like Operation 2, let A be the set of the pebbles outside S with distance at most xM from v. If there exists a dense vertex v (with $|A| \geq (2x + 2)M$), we do the following. We gather all pebbles of set A on vertex v. If the distance of v from at least one pebble in S is at most $(2x + 2)M$, we can use these pebbles in A to connect vertex v to set S. This way we add pebbles of A to set S by moving them along at most $xM + (2x + 2)M = (3x + 2)M$.

Similarly to Operation 2, if the distance of v from set S is more than $(2x + 2)M$, we can construct a path from a vertex w_1 with distance at most $(x + 1)M$ from vertex

v to a vertex w_2 with distance at most $(x+1)M$ from set S using pebbles outside S and A. Like before we can shift this path toward set S, and use the $(2x+2)M$ pebbles gathered on v to fill in the empty vertices of the path we are building from v to set S. The maximum movement in this case is at most $Max\{(3x+2)M, \lambda M+(2x+2)M\} = (3x+2)M$.

We now want to investigate the structure of the pebbles outside S in the optimal tree T. So we again delete all vertices from tree T that contain some pebbles of set S. We note that tree T is a spanning tree of the subgraph induced on the final positions of the pebbles in the optimum solution. Like before let $T'_1, T'_2, \ldots, T'_{k'}$ be the resulting subtrees of the forest we obtain after removing the above vertices from tree T.

Now we are ready to finish our algorithm in the third phase which is a non-iterative phase. We explain the main idea at first. If we knew which pebbles belong to each T'_i (the pebbles that move to vertices of T'_i in the optimum solution) for every $1 \leq i \leq k'$, we would do the following. For every tree T'_i we know that it either has diameter less than $2(\lambda-1)M$ or is a tree such that all its pebbles are close to its longest path. If it has small diameter, we can say that all its pebbles have distance at most $(x+2\lambda)M$ from some pebble in S. We prove this claim later in Lemma 4. About the second type trees, we can find a path connecting two vertices of T'_i with some specific properties. Then we can claim that every pebble in tree T'_i is close to some vertices of the path we find. This is just a raw idea without details. We will show how to distinguish between pebbles of different subtrees, and how to handle each tree (specially the second type trees).

At first we note that two adjacent pebbles in a subtree have distance at most $M+1+M = 2M+1$ from each other. So if we construct a graph with pebbles as its vertices, and connect each pair of pebbles that have distance at most $2M+1$ from each other, we can say that the pebbles of a subtree T'_i are in the same connected component in this graph. But there might be more than one subtree in a connected component. So we still have the distinguishing problem in a connected component.

The following trick helps us deal with this problem. We do the following operation to get rid of the small diameter subtrees. Note that the following operation is non-iterative; we do it once.

Operation 5. We mark all pebbles outside set S that has distance at most $(x+2\lambda)M$ from the current position of at least a pebble in S. After marking pebbles, we move all marked pebbles toward some pebbles in S with maximum movement at most $(x+2\lambda)M$.

Lemma 4. *Every pebble in a tree T'_i with small diameter (less than $2(\lambda-1)M$) is marked and inserted into set S.*

After Operation 5, we know that all small diameter subtrees are taken care of. Now we might be able to separate different subtrees with large diameter. There are two main issues here. The first one is that some pebbles of a tree T'_i with large diameter might be deleted. The second issue is that two pebbles in two different large diameter subtrees might have distance at most $2M+1$ (and therefore adjacent in the graph of pebbles we build).

Following we consider the graph of remaining pebbles, and show that both of these problems can be handled in a sophisticated way. Construct graph H with the remaining pebbles (outside S) as its vertices. And we connect two pebbles via an edge if its distance is at most $2M+1$. If this graph has multiple connected components, we treat each connected component separately. So we assume that this graph has only one connected component. At first we prove some properties of the remaining subtrees. This helps us understand the structure of the graph H.

Definition 2. *Define P_i to be the longest path of subtree T'_i. A subtree T'_i is called long tree, if P_i has length more than $2xM$. Let l'_i be the length of path P_i, and $q_{i,j}$ be the jth pebble of the path P_i for $1 \leq j \leq l'_i$. Also define $v'_{i,j}$ be the jth vertex of the path P_i. Note that there might be also some medium diameter trees other than small diameter and long trees.*

Following we prove that one of the last vertices of the longest paths of every remaining tree (with diameter at least $2(\lambda - 1)M$) is close to set S.

Lemma 5. *For every tree T'_i with diameter at least $2(\lambda - 1)M$, there exists a vertex v in tree T such that v has distance at most $10M$ from vertex v'_{i,l'_i}, the last vertex of the longest path in T'_i, and vertex v is also the final position of some pebble in S in the optimum solution. Therefore vertex v'_{i,l'_i} has distance at most $10M + M + xM = (x + 11)M$ from set S.*

Now we prove that the middle pebbles of long trees (which are the main parts of these trees) are not removed in Operation 5.

Lemma 6. *Consider a long tree T'_i. Pebble $q_{i,j}$ is unmarked (and therefore is not deleted) for all values of $xM + 1 \leq j \leq l'_i - xM$.*

Now that we have more information about the structure of the remaining subtrees, we can take care of them. We consider two cases. Note that H is the graph we constructed with the remaining pebbles as its vertices.

Case 1. Every pebble in H has distance at most $M+2xM+M+xM = (3x+2)M$ from some pebble in S. In this case we can move all pebbles of H and connect them to S. The problem is solved completely in this case and the maximum movement of these remaining pebbles would be at most $(3x+2)M$. Note that if there is no long tree among our trees, our problem will be solved in this case. Because we know that in a subtree, there exists a vertex v which is adjacent to the final position of some pebble p in S in the optimum solution (refer to the beginning of proof of Lemma 5). If the longest path of this subtree is at most $2xM$, it means that the current position of each pebble has distance at most $M+2xM$ from vertex v. We also know that it takes $M+xM$ edges to reach set S from vertex v because this subtree is obtained after the first four operations. The total distance is not more than $M + 2xM + M + xM = (3x + 2)M$. We conclude that if we have no long tree, the problem is solved in this case.

Case 2. As proved above, in this case we have some long trees. In Lemma 6, we proved that the middle parts of the longest paths of long trees do not get marked, and still exist in graph H. We just need to somehow add the pebbles of these middle parts of the long trees to set S. Because every pebble in a long tree has distance at most $4M$ from a pebble in the longest path in the tree. Each pebble in the longest path is either in the middle part of the path or has distance at most $M + xM + M = (x + 2)M$ from a pebble in the middle part. So every pebble in a long tree would be close (with distance at most a constant times M) to set S, if we add the pebbles in the middle parts of the longest paths of long trees to S. We also know that a tree that is not long has diameter at most $2xM$. This is enough to see that every pebble in a not long tree (medium tree) is close to set S already. So the main problem is finding a way to add pebbles of the middle parts of the longest paths of long trees into S.

Before starting we note that we still might have pebbles from several trees in our graph H. But we know that all middle pebbles of a long tree exist in the same connected component. In fact every pair of pebbles in graph H that are adjacent in tree T, are also

adjacent in H. We now show that there cannot be pebbles from three different trees in our connected graph H. There might be pebbles of two trees in H, but we show how to separate pebbles of these two trees from each other. The following lemma shows the limitations of the edges between pebbles of different trees. In fact, the following Lemma is the main reason that we are able to separate different subtrees.

Lemma 7. *If there is an edge in graph H between two pebbles $q \in T'_1$ and $q' \in T'_2$, we have that the distance between $v'_{1,1}$ and q is at most $11M$, and the distance between $v'_{2,1}$ and q' is also at most $11M$.*

Using Lemma 7, we can prove that there cannot be pebbles from three different trees in H as follows.

Lemma 8. *There can be pebbles from at most two trees in graph H.*

Now we know that there are pebbles from at most two trees in our graph H. We also know that all edges between pebbles of these two trees are close to the tail vertices of the longest paths of two trees. So we can get rid of all these edges by removing vertices around one of this tail vertices. Formally we do the following.

Let T'_1 and T'_2 be the two trees that contribute in H (this approach also works when there is only one tree). We can assume that we know vertex $v'_{1,1}$ because we can guess it (there are at most n possible guesses and one of them is correct).

We remove all pebbles of distance at most $11M$ from $v'_{1,1}$. We now have a graph H' with probably several connected components. Following we show that the middle parts of the long paths is safe. In fact we prove that we might remove at most the first $20M$ pebbles of the longest path, and we do not remove the pebbles $q_{1,20M+1}, q_{1,20M+2}, \ldots$. Otherwise we will have a dense vertex which is a contradiction.

Lemma 9. *After deleting pebbles with distance at most $11M$ from $v'_{1,1}$, we do not have any edge between pebbles of our two different trees. We also do not delete any of pebbles $v'_{1,20M+1}, v'_{1,20M+2}, \ldots$ if tree T'_1 is a long tree. And we do not delete pebbles $v'_{2,20M+1}, v'_{20M+2}, \ldots$ if tree T'_2 is a long tree.*

So we do not delete any pebble from middle parts of the long trees. We also removed a set of pebbles in Operation 4. But in Lemma 6, we proved that the middle parts of the longest paths of long trees survive. We conclude the following general lemma.

Lemma 10. *In the graph of remaining pebbles, graph H', pebbles $q_{1,xM+1}$, $q_{1,xM+2}$, $\ldots$, q_{1,l'_1-xM} exist and are in the same connected component for a long tree T'_1.*

So there might be at most two connected components containing the middle pebbles of longest paths of long trees. Remember these are the only parts we should handle, the rest of the pebbles are either close to set S or close to these two middle parts. In this part, we again treat pebbles of each connected component of graph H' separately. If all pebbles of a connected component in H' are close to set S, we can treat it like case 1 (we can move them directly toward set S). Otherwise this connected component has the middle pebbles of the longest path of either T'_1 or T'_2 (and not both of them for sure). Without loss of generality, assume that it has the middle pebbles of T'_1. So we know that all pebbles $q_{1,xM+1}, q_{1,xM+2}, \ldots, q_{1,l'_1-xM}$ are in the this connected component. We can assume that we know vertices $v'_{1,xM+1}$ and v'_{1,l'_1-xM} (we can guess, and there are at most n^2 possibilities). We know that there is a way to move some pebbles of this connected component to connect s to t with maximum movement at most

M. Using our algorithm for path movement problem, we can move some pebbles to connect these two vertices with maximum movement at most λM. We prove that all pebbles $q_{1,xM+1}, q_{1,xM+2}, \ldots, q_{1,l'_1-xM}$ are either used in the path we constructed, or are close to some pebble that we used in this path. At first we note that there is no edge between two pebbles from different trees. So all pebbles that we use are in the tree T'_1.

Lemma 11. *For every $xM+1 \leq i \leq l'_1 - xM - 2(\lambda-1)M$, pebble $q_{1,i}$ has distance at most $(2\lambda+20)M$ from either the starting position of one of the pebbles we used in the path we constructed from vertex $v'_{1,xM+1}$ to vertex v'_{1,l'_1-xM}, or one of the vertices of this path.*

We can gather all pebbles of our connected component on the path we constructed as follows.

Lemma 12. *Every pebble of our connected component in graph H', has distance at most $(x+2\lambda+3)M$ from some vertex of the path we constructed.*

Using Lemma 12, we can gather all pebbles of the connected component on our path. Now we have to move this path and connect it to S. Note that we can shift the path we constructed to make it connected to S. Vertex v'_{1,l'_1-xM} has distance at most xM from v'_{1,l'_1}. Vertex v'_{1,l'_1} has distance at most $(x+11)M$ from set S using Lemma 5. So we can shift the pebbles of this path along at most $xM+(x+11)M = (2x+11)M$ edges to make them connected to set S. So the maximum movement of all these parts (gathering and shifting) is at most $(x+2\lambda+3)M + (2x+11)M = (3x+2\lambda+14)M$ which is equal to $(8\lambda+80)M$. Now everything is connected to S, and no pebble is remained.

References

1. Stefano Basagni, Alessio Carosi, and Chiara Petrioli. *Heuristics for Lifetime Maximization in Wireless Sensor Networks with Multiple Mobile Sinks*. In *Proceedings of IEEE International Conference on Communications (ICC 2009)*, pages 1–6.
2. Stefano Basagni, Alessio Carosi, Chiara Petrioli, and Cynthia A. Phillips. *Coordinated and Controlled Mobility of Multiple Sinks for Maximizing the Lifetime of Wireless Sensor Networks*. to appear in ACM/Springer Wireless Networks (WINET).
3. Stefano Basagni, Alessio Carosi, Chiara Petrioli, and Cynthia A. Phillips. *Moving multiple sinks through wireless sensor networks for lifetime maximization*. In *Proceedings of IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2008)*, pages 523–526.
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Third Edition.
5. Erik D. Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, Shayan Oveis Gharan, Amin Sayedi-Roshkhar and Morteza Zadimoghaddam. *Minimizing movement. ACM Transactions on Algorithms*, 5(3), Article 30, July 2009. Preliminary version appeared at SODA 2007.
6. Erik D. Demaine, MohammadTaghi Hajiaghayi, and Daniel Marx. *Minimizing Movement: Fixed-Parameter Tractability.* In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009)*, pages 718-729.
7. Douglas B. West. *Introduction to Graph Theory*. Published by Prentice Hall, 2001.
8. Zachary Friggstad and Mohammad R. Salavatipour. *Minimizing movement in mobile facility location problems.* In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 357–366.
9. Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006.