Exact and approximate algorithms for movement problems on (special classes of) graphs[*]

Davide Bilò[1], Luciano Gualà[2], Stefano Leucci[3], and Guido Proietti[3,4]

[1] Dipartimento di Scienze Umanistiche e Sociali, University of Sassari, Italy.
[2] Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Italy.
[3] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila, Italy.
[4] Istituto di Analisi dei Sistemi ed Informatica, CNR, Rome, Italy.
davidebilo@uniss.it, guala@mat.uniroma2.it, {stefano.leucci,guido.proietti}@univaq.it

Abstract. When a large collection of objects (e.g., robots, sensors, etc.) has to be deployed in a given environment, it is often required to plan a coordinated motion of the objects from their initial position to a final configuration enjoying some global property. In such a scenario, the problem of minimizing some function of the distance travelled, and therefore energy consumption, is of vital importance. In this paper we study several motion planning problems that arise when the objects must be moved on a graph, in order to reach certain goals which are of interest for several network applications. Among the others, these goals include broadcasting messages and forming connected or interference-free networks. We study these problems with the aim of minimizing a number of natural measures such as the average/overall distance travelled, the maximum distance travelled, or the number of objects that need to be moved. To this respect, we provide several approximability and inapproximability results, most of which are tight.

1 Introduction

In many practical applications a number of centrally controlled objects need to be moved in a given environment in order to complete some task. Problems of this kind often occur in robot motion planning where we seek to move a set of robots from their starting position to a set of ending positions such that a certain property is satisfied. For example, if the robots are equipped with a short range communication device we might want to move them so that a message originating from one of the robots can be routed to all the others. If the robots' goal is to monitor a certain area we might want to move them so that they are not too close to each other. Other interesting problems include gathering (placing robots next to each other), monitoring of traffic between two locations, building interference-free networks, and so on. To make things harder, objects to be moved are often equipped with a limited supply of energy. Preserving energy is a critical problem in ad-hoc networking, and movements are expensive. To prolong the lifetime of the objects we seek to minimize the energy consumed during

---

movements and thus the distance travelled. Sometimes, instead, movements are cheap but before and/or after an object moves it needs to perform expensive operations. In this scenario we might be interested in moving the minimum number of objects needed to reach the goal.

In this paper, we assume the underlying environment is actually a *network*, which can be modelled as an undirected graph G, and the moving objects are centrally controlled *pebbles* that are initially placed on vertices of G, and that can be moved to other vertices by traversing the graph edges. To this respect, we study several movement planning problems that arise by various combinations of final positioning goals and movement optimization measures. In particular, we focus our study on the scenarios where we want the pebbles to be moved to a *connected subgraph* (CON), an *independent set* (IND), or a *clique* (CLIQUE) of G, while minimizing either the *overall movement* (SUM), the *maximum movement* (MAX), or the *number of moved pebbles* (NUM). We also give some preliminary results on the problem of moving the pebbles to an *s-t-cut*, i.e., a set of vertices whose removal makes two given vertices s, t disconnected (*s-t*-CUT) while minimizing the above measures.

We will denote each of the above problems with ψ-c, where ψ represents the goal to be achieved and c the measure to be minimized. For a more rigorous definition of the problems we refer the reader to Section 2.

Related work. Although movement problems were deeply investigated in a distributed setting (see [15] for a survey), quite surprisingly the centralized counterpart has received attention from the scientific community only in the last few years.

The first paper which defines and studies these problems in this latter setting is [7]. In their work, the authors study the problem of moving the pebbles on a graph G of n vertices so that their final positions form a *connected component*, a *path* (directed or undirected) *between two specified nodes*, an *independent set*, or a *matching* (two pebbles are matched together if their distance is exactly 1).

Regarding connectivity problems, in [7] the authors show that all the variants are hard and that the approximation ratio of CON-MAX is between 2 and $O(1 + \sqrt{k/c^*})$, where k is the number of pebbles and c^* denotes the measure of an optimal solution. This result has been improved in [4], where the authors show that CON-MAX can be approximated within a constant factor. In [7] it is also shown that CON-SUM and CON-NUM are not approximable within $O(n^{1-\epsilon})$ (for any positive ϵ) and $o(\log n)$, respectively, while they admit approximation algorithms with ratios of $O(\min\{n \log n, k\})$ and $O(k^\epsilon)$, respectively. Moreover, the authors also provide an exact polynomial-time algorithm for CON-MAX on trees.

Concerning independency problems, in [7] the authors remark that it is NP-hard even to find any feasible solution on general graphs since it would require to find an independent set of size at least k. This clearly holds for all three objective functions.

For this reason, they study an Euclidean variant of these problems where pebbles have to be moved on a plane so that their pairwise distances are strictly greater than 1. In this case, the authors provide an approximation algorithm that guarantees an additive error of at most $1 + 1/\sqrt{3}$ for IND-MAX, and a polynomial time approximation scheme for IND-NUM.

More recently, in [10], a variant of the classical facility location problem has been studied. This variant, called *mobile facility location*, can be modelled as a movement problem and is approximable within $(3 + \epsilon)$ (for any constant $\epsilon > 0$) if we seek to minimize the total movement [1], while the variant where the maximum movement has to be minimized admits a tight 2-approximation [7,10]. Moreover, as it is frequent in the practice to have a small number of pebbles compared to the size of the environment (i.e., the vertices of the graph), the authors of [8] turn to study fixed-parameter tractability. They show a relation between the complexity of the problems and their *minimal configurations* (sets of final positions of the pebbles that correspond to feasible solutions, such that any removal of an edge makes them unacceptable). Finally, we mention that in [2] it was considered a set of vertex-to-vertex motion planning problems in a simple polygon, with the aim of forming final configurations enjoying some sort of *visual connectivity* among the pebbles.

Our results. We start by studying connectivity motions problems in the case where pebbles move on a tree, and we devise two polynomial-time dynamic programming algorithms for CON-SUM and CON-NUM. These algorithms complement the already known polynomial-time algorithm for CON-MAX on trees shown in [7].

Then, we study independency motion problems on graphs where a *maximum independent set* (and thus a feasible solution for the corresponding motion problem) can be computed in polynomial time. This class of graphs includes, for example, perfect and claw-free graphs. More precisely, we show that IND-MAX and IND-SUM are NP-hard even on bipartite graphs (which are known to be perfect graphs [5]). Moreover, we devise three exact polynomial-time algorithms: one for solving IND-MAX on paths, and the other two for solving IND-SUM and IND-NUM on trees, respectively. Moreover, we devise a polynomial-time approximation algorithm for IND-MAX which is optimal unless an additive term of 1 (this is clearly tight).

Concerning the problem of moving pebbles towards a clique of a general graph, we prove that all the three variants are NP-hard. Then, we provide an approximation algorithm for CLIQUE-MAX which is optimal unless an additive term of 1 (this result is clearly tight). Moreover, we show that both CLIQUE-SUM and CLIQUE-NUM are approximable within a factor of 2, but they are not approximable within a factor better than $10\sqrt{5} - 21 > 1.3606$, unless P = NP. If the *unique game conjecture* [13] is true, then both problems are not approximable within a factor better than 2 and the provided approximation algorithms are tight. These results are obtained by showing

3

a non-trivial relation with the *minimum vertex cover* problem. We also show that an exact solution for Clique-Num can be computed in polynomial time on every class of graphs for which finding a *maximum-weight clique* requires polynomial time (these classes of graphs also include perfect and claw-free graphs).

Finally, we present a strong inapproximability results of $\Omega(n^{1-\epsilon})$ (for any $\epsilon > 0$) for s-t-Cut-Max and s-t-Cut-Sum, unless $\mathsf{P} = \mathsf{NP}$, along with two approximation algorithms. The approximation algorithm for s-t-Cut-Max is essentially tight, while we show that any constant-factor approximation for s-t-Cut-Num would imply a tight approximation for s-t-Cut-Sum.

The paper is organized as follows: in Section 2 we provide a formal definition of our problems, while in Sections 3–6 we give our results for Con, Ind, Clique, and s-t-Cut, respectively (for a summary of the state of the art of the studied problems, along with the results presented in this paper, see Table 1). Finally, Section 7 concludes the paper.

	Max	Sum	Num
Con	G: $2 \le \rho = O(1)$ [4,7] T: polynomial [7]	G: $\rho = \Omega(n^{1-\epsilon})$ [7] $\rho = O(\min\{n \log n, k\})$ [7] **T: polynomial**	G: $\rho = \Omega(\log n)$ [7] $\rho = O(k^\epsilon)$ [7] **T: polynomial**
Ind	G: NP-hard [7] **IS: $c^* + 1$, $\rho \le 2$** **B: $\rho \ge 2$** **P: polynomial**	G: NP-hard [7] **B: NP-hard** **T: polynomial**	G: NP-hard [7] **T: polynomial**
Clique	**G: NP-hard** $c^* + 1$	**G: $10\sqrt{5} - 21 \le \rho \le 2$**	**G: $10\sqrt{5} - 21 \le \rho \le 2$** **MWC: polynomial**
s-t-Cut	**G: $\rho = \Omega(n^{1-\epsilon})$** $\rho \le d$	**G: $\rho = \Omega(n^{1-\epsilon})$** $\rho \le k \cdot d$	**G: ρ-apx $\implies$ ($\rho \cdot d$)-apx** **for s-t-Cut-Sum**

Table 1. Known and new (in bold) results for the various motion problems on general graphs (G), bipartite graphs (B), graphs on which a maximum independent set or a maximum-weight clique can be computed in polynomial time (IS, MWC), trees (T), and paths (P). With n and d we denote the number of vertices and the diameter of G, respectively, while k denotes the number of pebbles, ρ denotes the best approximation ratio for the corresponding problem, and finally c^* is the measure of an optimal solution. Notice that for independency problems on general graphs it is NP-hard even to find any feasible solution. All the inapproximability results hold under the assumption that $\mathsf{P} \ne \mathsf{NP}$.

2 Formal definitions

A pebble motion problem, denoted by ψ-c, is an optimization problem whose instances consist of a loop-free connected undirected graph $G = (V(G), E(G))$ on n nodes, a set $P = [k] = \{1, \ldots, k\}$ of *pebbles*, a function $\sigma : P \to V(G)$ that assigns each pebble to a *start vertex* of G, and a boolean predicate $\psi : 2^{V(G)} \to \{\text{true}, \text{false}\}$ that assigns a truth value to every possible subset of vertices of G.

A (feasible) solution is a function $\mu : P \to V(G)$ that maps each pebble to an *end vertex* of G (in other words, *moves* a pebble from its start to its end position) such that $\psi(\mu[P])$ is true, where $\mu[P]$ denotes the image of P under μ. Notice that, in general, it is not required for σ or μ to be injective and thus we allow more than one pebble to be placed on the same vertex. In the rest of the paper, we will assume that a pebble moving from a vertex u to a vertex v always uses a *shortest path* in G between u and v, say $\pi_G(u, v)$. Moreover we denote by $d_G(u, v)$ the length of such a path. Finally, $c(\mu) \in \mathbb{N}_0$ is a measure function that assigns a non-negative integer to each feasible solution (i.e., to each set of moves). A solution μ^* that minimizes c is said to be *optimal*.

In the following, we will study some of the movement problems that arise from the different choices of predicates and measures. In particular, we will consider the following predicates:

Connectivity: $\text{CON}(U)$ is true if and only if the subgraph of G induced by the set of vertices $U \subseteq V(G)$ is connected;

Independency: $\text{IND}(U)$ is true if and only if $U \subseteq V(G)$ is an independent set of G of size k, i.e., there is at most one pebble per vertex and no two pebbles are on adjacent vertices;

Clique: $\text{CLIQUE}(U)$ is true if and only if $U \subseteq V(G)$ induces a clique in G, i.e., for each pair u, v of distinct vertices in U there exists the edge $(u, v) \in E(G)$;

s-t-Cut: Given $s, t \in V(G)$ with $s \neq t$, then s-t-$\text{CUT}(U)$ is true if and only if $s \notin U$, $t \notin U$ and $U \subseteq V(G)$ is an s-t-cut (i.e., there exists no path between s and t in the graph induced by the vertices in $V(G) \setminus U$);

and the following measures:

Overall movement: The sum of the distances travelled by pebbles has to be minimized: every pebble $p \in P$ moves from its starting vertex $\sigma(p)$ to his end vertex $\mu(p)$, so the overall distance is $\text{SUM}(\mu) = \sum_{p \in P} d_G(\sigma(p), \mu(p))$;

Maximum movement: We seek for a solution that minimizes the maximum distance travelled by a pebble, i.e., the measure $\text{MAX}(\mu) = \max_{p \in P} d_G(\sigma(p), \mu(p))$;

Number of moved pebbles: We aim to minimize the number of pebbles that need to be moved from their starting positions. The associated measure is $\text{NUM}(\mu) = |\{p \in P : \sigma(p) \neq \mu(p)\}|$.

3 Connectivity motion problems

In this section we describe two polynomial-time algorithms for solving on trees CON-SUM and CON-NUM, respectively. In this way we complement the result provided in [7] for CON-MAX on trees.

3.1 Solving CON-SUM on trees

Our dynamic-programming algorithm relies on the following property of optimal solutions:

Lemma 1. *In any optimal solution μ^* for an instance of* CON-SUM *on trees, there exists no edge that is traversed in opposite directions by pebbles.*

Proof. Let $\pi_G(u, v)$ denote a shortest path in G between the vertices u and v. Suppose by contradiction that there exists an optimal solution μ^*, an edge $(x, y) \in E(G)$, and two pebbles $p, q \in P$ such that p moves through the path $\pi_G(\sigma(p), x) \cup \{(x, y)\} \cup \pi_G(y, \mu^*(p))$ and q moves through the path $\pi_G(\sigma(q), y) \cup \{(y, x)\} \cup \pi_G(x, \mu^*(q))$. Consider the solution μ' obtained from μ^* by swapping the final positions for p and q, i.e., $\mu'(p) = \mu^*(q)$ and $\mu'(q) = \mu^*(p)$. Clearly $\mu'[P] = \mu^*[P]$, therefore μ' is feasible. Moreover $c(\mu') < c(\mu^*)$ as we can move p through the path $\pi_G(\sigma(p), x) \cup \pi_G(x, \mu'(p))$ and q through the path $\pi_G(\sigma(q), y) \cup \pi_G(y, \mu'(q))$, thus saving 2. $\square$

The algorithm first guesses a vertex $r \in V(G)$ such that there exists an optimal solution that places a pebble on r, then roots the tree G at r to obtain a rooted tree G_r, and finally considers all the subtrees of G_r in a bottom-up fashion.

For a given subtree T_u of G_r rooted at the vertex u, let us denote by $\eta(u)$ the number of pebbles placed on $V(T_u)$ w.r.t. σ. When the subtree T_u is examined, we consider an auxiliary problem. In this problem we want to place exactly $j \leq k$ of the pebbles on the vertices of T_u in order to satisfy the following properties:

(P1) the subgraph of T_u induced by the final positions of the pebbles must be connected;

(P2) if $j \geq 1$ then at least one pebble must be placed on u.

Moreover, if $j < \eta(u)$ we want to move the $\eta(u) - j$ exceeding pebbles to the parent of u (and thus outside T_u). In a similar manner, if $j > \eta(u)$ then the $j - \eta(u)$ missing pebbles are to be moved into T_u from the parent of u, where we assume they are initially placed. We point out that, by Lemma 1, we do not need to consider the case where some pebbles move into T_u while others move out of T_u.

We will denote by $\texttt{Opt}[u, j]$ the cost of the optimal movement for this auxiliary problem. Notice that, in $\texttt{Opt}[u, j]$, we are accounting for the cost of traversing all the edges of T_u plus the edge from u to its parent. To solve the original problem we need to find a solution corresponding to $\texttt{Opt}[r, k]$. Clearly as $k = \eta(r)$ we do not have exceeding or missing pebbles, in this case. We now show how to combine these auxiliary problems.

If u is a leaf of G_r then $\texttt{Opt}[u, j] = |\eta(u) - j|$. Otherwise, if u is not a leaf, we can distinguish two cases: $j = 0$ and $j > 0$. If $j = 0$ then no pebble can be placed on u or in any descendant of u, therefore all the pebbles must first be moved towards u

6

and then to the parent of u. Let $v_1, \ldots, v_\ell$ be the set of children of u in G_r, we have:
$\mathtt{Opt}[u, 0] = \eta(u) + \sum_{i=1}^{\ell} \mathtt{Opt}[v_i, 0]$.

Otherwise, if $j > 0$, we can move any number of pebbles between 1 and j to u, and place the remaining pebbles on the subtrees rooted at the children of u. Therefore we have:

$$\mathtt{Opt}[u, j] = |\eta(u) - j| + \min_{\substack{0 \leq j_1, \ldots, j_\ell < j \\ \sum_{i=1}^{\ell} j_i < j}} \left\{ \sum_{i=1}^{\ell} \mathtt{Opt}[v_i, j_i] \right\}.$$

Notice that the minimum considers all the possible ways for distributing less than j pebbles on the subtrees, i.e., all the vectors $(j_i)_i$ of ℓ elements whose sum is less than j.

We now argue on the fact that, despite the number of such vectors can be exponential on j, the minimum can be found in polynomial time.

This can again be done by using dynamic programming: let $\mathtt{Min}[i, h]$ denote the minimum cost of placing h pebbles in the first i subtrees. Clearly when $i = 1$ we have $\mathtt{Min}[1, h] = \mathtt{Opt}[v_1, h]$, while for $i > 1$ the following holds:

$$\mathtt{Min}[i, h] = \min_{0 \leq z \leq h} \left\{ \mathtt{Min}[i - 1, z] + \mathtt{Opt}[v_i, h - z] \right\}.$$

Therefore, the equation for $\mathtt{Opt}[u, j]$ can be rewritten as:

$$\mathtt{Opt}[u, j] = |\eta(u) - j| + \min_{0 \leq i < j} \mathtt{Min}[\ell, i].$$

Notice how this way of distributing the pebbles is general and does not depend on the specific movement problem: in fact, it can be used every time we are interested in minimizing the cost of distributing a number of items in a set of bins if, for each bin, we incur a cost that depends on the number of items placed therein.

The time required to compute a specific $\mathtt{Opt}[u, j]$ is $O(\ell \cdot j) = O(\ell \cdot k)$. As the sum of the ℓ-values over all the vertices is $n - 1$, the time needed to compute $\mathtt{Opt}[v, j]$ for a fixed j and all $v \in V$ is $O(n \cdot k)$. It follows that all the possible subproblems can be solved in time $O(n \cdot k^2)$.

As we have to guess the vertex r, a naïve strategy would be repeating the above procedure n times, one for each vertex of G. This would require an overall time of $O(n^2 \cdot k^2)$. We can do better by using a more sophisticated approach: consider a *centroid*[5] v of G, and notice that either there exists an optimal solution that places a pebble on v, or every optimal solution places all the pebbles on a single connected component of $G - v$. We first apply the above algorithm using v as the root and then

---

[5] A centroid of a tree is a vertex whose removal minimizes the maximum number of nodes over all the trees of the resulting forest. Notice that each tree of the forest has at most half of the vertices of the original tree, and that a centroid can be easily found in linear time.

we proceed recursively on the trees of the forest $G - v$ (each of which has at most half of the vertices). More precisely, for every subtree T of $G - v$, rooted at v', we recursively solve an instance consisting of the tree T where all the pebbles in $V(T)$ are left unmoved and all pebbles not in $V(T)$ have been moved to v'. This movement cost, i.e., $\sum_{p \in P : \sigma(p) \notin V(T)} d(\sigma(p), v')$, is then added to the measure of the solution returned by the recursive call. Among all the computed solutions we choose the cheapest one.

By doing so, we are able to reduce the computational complexity to $O(n \cdot k^2 \log n)$. Indeed, the recurrence relation describing the running time of the algorithm is $T(n) = \sum_{n_j} T(n_j) + O(n \cdot k^2)$, where $n_j \leq \frac{n}{2}$ denotes the number of vertices of the j-th subtree of $G - v$. Clearly, the depth of the recursion is $O(\log n)$ while the amount of work on each level of the recursion-tree is $O(n \cdot k^2)$.

Once the value of the optimal solution μ^* has been found, it is not too hard to see that the optimal solution itself can be reconstructed by proceeding in a bottom-up fashion, while keeping track of both the pebbles that move out of each subtree and the position where missing pebbles are to be placed.

To summarize, we have the following:

Theorem 1. CON-SUM *on trees can be solved in* $O(n \cdot k^2 \log n)$ *time.*

3.2 Solving CON-NUM on trees

The algorithm is similar to the one for CON-SUM: we guess a vertex $r \in V(G)$ such that there exists an optimal solution that places a pebble on r, then we root the tree G at r (call G_r the rooted tree) and we consider all the subtrees of G_r in a bottom-up fashion.

Let $\varphi(u) = |\{p \in P : \sigma(p) = u\}|$ be the number of pebbles whose initial position is the vertex $u \in V(G)$.

As before, when the subtree T_u (rooted at the vertex u) of G_r is examined, we consider an auxiliary problem where we want to place exactly $j \leq k$ pebbles on the vertices of T_u in order to satisfy the properties (P1) and (P2).

We will measure the cost of a solution for this auxiliary problem by examining the number of pebbles placed on each vertex of T_u. Removing pebbles from a vertex costs nothing, while placing a pebble on a vertex costs 1 if it comes from a different vertex. A way to visualize this auxiliary problem is to imagine the tree T_u where no pebbles have been placed and a pool of j pebbles to be distributed on its vertices. Each vertex v of T_u can hold up to $\varphi(v)$ pebbles for free, while each additional pebble placed on v increases the overall cost by 1.

We will denote by $\texttt{Opt}[u, j]$ the cost of the optimal movement for this auxiliary problem. To solve the original problem we need to find a solution corresponding to $\texttt{Opt}[r, k]$. We now show how to combine these auxiliary problems.

If u is a leaf of G_r then we have:

$$\texttt{Opt}[u, j] = \begin{cases} 0 & \text{if } j \leq \varphi(u); \\ j - \varphi(u) & \text{if } j > \varphi(u). \end{cases}$$

Otherwise, if u is not a leaf, we can either place some pebbles on u and the others on the subtrees rooted at its children ($j \geq 1$), or place no pebble at all in the whole subtree rooted at u ($j = 0$). We call z the number of pebbles that are to be placed on u.

If $j = 0$ we have $\texttt{Opt}[u, j] = 0$, otherwise:

$$\texttt{Opt}[u, j] = \min_{1 \leq z \leq j} \left\{ \max\{z - \varphi(u), 0\} + \min_{\substack{0 \leq j_1, \ldots, j_\ell \leq j - z \\ \sum_{i=1}^{\ell} j_i = j - z}} \left\{ \sum_{i=1}^{\ell} \texttt{Opt}[v_i, j_i] \right\} \right\}$$

where $v_1, \ldots, v_\ell$ are the children of u in G_r.

As before, using the already shown dynamic-programming approach to optimally distributing the pebbles on the subtrees, we can find the values of $\texttt{Opt}[u, j]$ for a fixed j and all $v \in V$, in $O(n \cdot k)$ time. Therefore the time required to compute all $\texttt{Opt}$ values for a single root r is $O(n \cdot k^2)$ and the measure of the best solution is found in $\texttt{Opt}[r, k]$. As for CON-SUM, it is not necessary to run the algorithm for all roots $r \in V(G)$ but we can choose a centroid v of G as starting root and then proceed recursively on the trees of $G - v$.

To summarize, we have the following:

Theorem 2. CON-NUM *on trees can be solved in* $O(n \cdot k^2 \log n)$ *time.*

4   Independency motion problems

In this section we focus on independency motion problems. First, we give a better characterization of the hardness of IND-MAX and IND-SUM (depending on the input graph), and then we show some positive results for our considered variants on paths and trees. Since if $k \geq n$ there is no feasible solution, we will consider only instances where $k < n$.

4.1   Hardness of IND-MAX and IND-SUM on bipartite graphs

As we already pointed out, for independency problems on general graphs it is NP-hard even to find any feasible solution since it would require to find an independent set of size at least k. Nevertheless, one may wonder whether independency motion problems are tractable on instances on which a maximum independent set can be found in polynomial time. We provide a negative answer to this question, at least for IND-MAX and IND-SUM, by showing the following
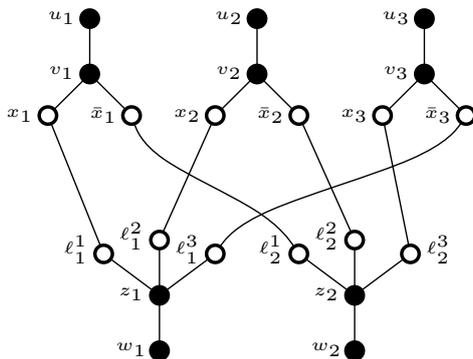
Fig. 1. Instance of IND-MAX and of IND-SUM corresponding to the formula $(\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$. Pebbles are placed on black vertices.

Theorem 3. IND-MAX *and* IND-SUM *are* NP-*hard on bipartite graphs.*

Proof. We will show a polynomial reduction from 3-SAT to the decisional versions of IND-MAX and IND-SUM. Recall that 3-SAT is the problem of deciding whether a formula f in conjunctive normal form with three literals per clause is satisfiable. Let $X = \{x_1, \ldots, x_\tau\}$ be the set containing the variables of f, and let m be the number of clauses. We will denote the i-th literal of the j-th clause with ℓ_j^i.

Given an instance f for 3-SAT we construct an instance for IND-MAX and for IND-SUM in the following manner:

- For each variable $x_i \in X$ create a star with 3 leaves labelled $u_i, x_i, \bar{x}_i$ and label the internal node v_i. Place one pebble on u_i and one on v_i.
- For each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ create a star with 4 leaves labelled $\ell_j^1, \ell_j^2, \ell_j^3, w_j$ and label the internal node z_j. Place one pebble on w_j and one on z_j.
- For each literal ℓ_j^i of f let x_s be the corresponding variable; then, if ℓ_j^i is asserted add an edge between the two nodes labelled ℓ_j^i and $\bar{x}_s$, otherwise add an edge between the two nodes labelled ℓ_j^i and x_s.

Let G be the resulting graph, P the set containing the $2 \cdot (\tau + m)$ placed pebbles, and σ the function that maps each pebble to its starting position (see Figure 1). Notice that G is bipartite as we can partition the vertices into two sets $A = \{v_i : 1 \leq i \leq \tau\} \cup \{\ell_j^i : 1 \leq i \leq 3 \wedge 1 \leq j \leq m\} \cup \{w_j : 1 \leq j \leq m\}$ and $B = V(G) \setminus A$ such that no edge of G has both its endpoints in the same set.

We claim that there exists an assignment that satisfies f if and only if the optimal solution μ for the instance of IND-MAX (resp., IND-SUM) has measure at most 1 (resp., $\tau + m$).

Suppose the existence of an assignment $\theta^* : X \to \{\text{true}, \text{false}\}$ that satisfies f. Then we move a pebble starting on vertex v_i to the vertex labelled x_i if $\theta^*(x_i) = \text{true}$,

10

or to the vertex labelled $\bar{x}_i$ if $\theta^*(x_i) = \text{false}$. Moreover, for each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ of f at least one literal ℓ_j^i must be true w.r.t. θ^*. This implies that the vertex labelled ℓ_j^i is adjacent only to z_j and to a vertex x_s where no pebble has been placed. We then move the pebble initially placed on vertex z_j to the vertex ℓ_j^i.

The resulting configuration of pebbles is an independent set for G and each pebble has been moved to a node adjacent to its starting position. This implies that the maximum movement is 1 and that the overall distance travelled by pebbles is $\tau + m$.

Conversely, suppose the existence of an optimal solution μ^* for the instance of IND-MAX (resp., IND-SUM) that has measure equal to 1 (resp., $\tau + m$). Notice that for every pebble p initially placed on a vertex labelled v_i, $\mu^*(p) \in \{x_i, \bar{x}_i\}$ must hold, otherwise either μ^* would be unfeasible or $c(\mu^*)$ would be greater than 1 (resp., $\tau + m$, since $\tau + m$ pebbles need to be moved). Similarly for every pebble p initially placed on a vertex labelled z_j, $\mu^*(p) \in \{\ell_1^j, \ell_2^j, \ell_3^j\}$ must hold.

We construct an assignment θ for the variables in the following manner: if there is a pebble on the vertex labelled x_i we set $\theta(x_i) = \text{true}$, otherwise there must be a pebble on $\bar{x}_i$ and we set $\theta(x_i) = \text{false}$.

We now show that θ is, indeed, an assignment that satisfies f. For each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ the pebble placed on z_j has been moved to a vertex ℓ_j^i corresponding to one of the three literals. This implies that no pebble has been moved to the unique vertex in $\cup_{i=1}^t \{x_i, \bar{x}_i\}$ that is adjacent to vertex ℓ_j^i. The above implies that the variable corresponding to literal ℓ_i^j has been set to the value that satisfies ℓ_i^j and thus the whole clause is satisfied. □

Apparently, the above technique cannot be straightforwardly adapted to IND-NUM, and so we leave this as an interesting open problem.

4.2 Approximability of IND-MAX

Actually, as shown in Theorem 3, IND-MAX is hard already when the cost of an optimal solution is 1. This immediately implies the following:

Corollary 1. IND-MAX *on bipartite graphs is not approximable in polynomial time within a factor of $2 - \epsilon$ for any positive ϵ, unless* $\mathsf{P} = \mathsf{NP}$.

We now show that this bound is tight, by providing a polynomial-time solution, which is optimal unless an additive term of 1, to IND-MAX on any class of graphs where a maximum independent set can be found in polynomial time, e.g., perfect graphs (which include bipartite graphs), interval graphs, and claw-free graphs.

Given a graph H and a subset of vertices $A \subseteq V(H)$ we will denote the open neighbourhood of A by $N_H(A) = \{v \in V(H) : \exists u \in A \text{ s.t. } (u,v) \in E(H)\}$. Moreover we will denote the closed neighbourhood of A by $N_H[A] = A \cup N_H(A)$.

Let U^* be a maximum independent set of G, the following lemma holds:

Lemma 2. *For each independent set U of G it is true that $|U^* \cap N_G[U]| \geq |U|$.*

Proof. By contradiction, let $|U^* \cap N_G[U]| < |U|$ then $U' = (U^* \setminus N_G[U]) \cup U = (U^* \setminus (U^* \cap N_G[U])) \cup U$ is an independent set of G and $|U'| > |U^*|$. $\qquad\square$

To prove the next lemma we use the following well known result:

Theorem 4 (Hall's Matching Theorem [11]). *Let $H = (V_1 \cup V_2, E)$ be a bipartite graph. There exists a matching of size $|V_1|$ on H iff $|A| \leq |N_H(A)|$, $\forall A \subseteq V_1$.*

Lemma 3. *For each independent set U of G, there exists an injective function $f : U \rightarrow U^*$ such that $d_G(u, f(u)) \leq 1$.*

Proof. Construct the bipartite graph $H = (U \cup U^*, E)$ where all vertices of U are considered to be distinct from the ones in U^* and $E = \{(u, v) \in U \times U^* : v \in N_G[\{u\}]\}$. Notice that, by construction, if two vertices u, v are adjacent in H either they are the same vertex or they are adjacent in G, i.e., $d_G(u, v) \leq 1$.

Now, Lemma 2 implies that, for every $A \subseteq U$, we have $|N(A)| = |U^* \cap N_G[A]| \geq |A|$. Hence, from Hall's Matching Theorem, there is a matching of size $|U|$ on H (and thus the function f exists). $\qquad\square$

We are now ready to prove:

Theorem 5. *There exists a polynomial-time algorithm for* IND-MAX *which, for every class of graphs where the maximum independent set can be found in polynomial time, computes a solution $\tilde{\mu}$ such that $c(\tilde{\mu}) \leq c^* + 1$ where c^* is the measure of an optimal solution.*

Proof. The algorithm computes a maximum independent set U^* of G then, if $k > |U^*|$ it reports infeasibility, otherwise it optimally moves the pebbles towards (a subset of) the vertices of U^*. To do that, it proceeds as follows: for every value of z from 0 to $n-1$, it computes a solution S_z for the maximum matching problem on the auxiliary bipartite graph $H = (A \cup U^*, E)$, where each vertex in A is associated with a pebble, and $(p \in A, v \in U^*) \in E$ if and only if $d_G(\sigma(p), v) \leq z$.

Let $\tilde{z}$ be the first value of z such that $|S_z| = k$, i.e., all the pebbles have been matched. For every pebble $p \in P$ set $\tilde{\mu}(p) = v$, where v is the only vertex such that $(p, v) \in S_{\tilde{z}}$, and return $\tilde{\mu}$. Clearly $c(\tilde{\mu}) = \tilde{z}$. Let μ^* be an optimal solution to IND-MAX and let $U = \mu^*[P]$. By Lemma 3 there exists an injective function f that maps every vertex of the independent set U to an adjacent vertex of U^*. Thus, for every $p \in P$ we have $d_G(\sigma(p), \mu^*(p)) + d_G(\mu^*(p), f(\mu^*(p))) \leq c^* + 1$, and therefore there exists a way to place all the pebbles on vertices of U^* while travelling a maximum distance of at most $c^* + 1$. This implies $\tilde{z} \leq c^* + 1$. $\qquad\square$

4.3 Independency motion problems on trees and paths

Concerning independency motion problems on trees, we are able to devise two dynamic-programming algorithms for IND-SUM and IND-NUM, respectively. We note that it remains open to establish whether IND-MAX on trees can be solved in polynomial time. We will, however, devise an efficient algorithm for solving IND-MAX on paths. The details of these algorithm are presented in the following subsections.

4.3.1 Solving IND-SUM on trees.

Our dynamic-programming algorithm relies on the property of optimal solutions shown by Lemma 1 that is valid also for IND-SUM.

The algorithm first roots the tree G at an arbitrary vertex $r \in V(G)$ to obtain the rooted tree G_r, then considers all the subtrees of G_r in a bottom-up fashion.

Here, the auxiliary problem we consider when a subtree T_u of G rooted at vertex u is examined is that of placing exactly $j \leq k$ pebbles on the vertices of T_u such that their final positions induce an independent set of G (and each pebble is placed on different vertex). Remind that $\eta(u)$ denotes the number of pebbles placed on $V(T_u)$. As for the connectivity problems, if $j < \eta(u)$ we want to move the $\eta(u) - j$ exceeding pebbles to the parent of u (and thus outside T_u). In a similar manner, if $j > \eta(u)$, then the $j - \eta(u)$ missing pebbles are to be moved into T_u from the parent of u, where we assume they are initially placed.

We will denote by $\mathtt{Opt}[u, j]$ the cost of the optimal movement for this auxiliary problem. Notice that, in $\mathtt{Opt}[u, j]$, we are accounting for the cost of traversing all the edges of T_u plus the edge from u to its parent. To solve the original problem we need to find a solution corresponding to $\mathtt{Opt}[r, k]$. Clearly as $k = \eta(r)$ we do not have exceeding or missing pebbles, in this case. We now show how to combine these auxiliary problems.

Let $\mathtt{Opt}^+[u, j]$ (resp., $\mathtt{Opt}^-[u, j]$) with $u \in V(G)$ and $0 \leq j \leq n$ be the value of an optimal solution to the auxiliary problem where exactly one pebble must be placed on u (resp., no pebble can be placed on u). Notice that $\mathtt{Opt}[u, j] = \min\{\mathtt{Opt}^+[u, j], \mathtt{Opt}^-[u, j]\}$. We will say that infeasible solutions have cost $+\infty$.

If u is a leaf we clearly have:

$$\mathtt{Opt}^+[u, j] = \begin{cases} |\eta(u) - 1| & \text{if } j = 1; \\ +\infty & \text{if } j = 0 \text{ or } j \geq 2. \end{cases} \qquad \mathtt{Opt}^-[u, j] = \begin{cases} \eta(u) & \text{if } j = 0; \\ +\infty & \text{if } j \geq 1. \end{cases}$$

If u is not a leaf then we can either place a pebble on u or not. If we do place it, the corresponding optimal value is:

$$\mathtt{Opt}^+[u, j] = \begin{cases} +\infty & \text{if } j = 0; \\ |\eta(u) - j| + \min_{\substack{0 \leq j_1, \ldots, j_\ell < j \\ \sum_{i=1}^{\ell} j_i = j-1}} \left\{ \sum_{i=1}^{\ell} \mathtt{Opt}^-[v_i, j_i] \right\} & \text{if } j \geq 1, \end{cases}$$

13

where $v_1, \ldots, v_\ell$ is the set of children of u in G_r. If j is at least 1 we place a pebble on u and then consider all the possible ways of placing a total of $j-1$ pebbles on the subtrees rooted at the children of u, but not on the children themselves.

If we do not place a pebble on u, then the optimal value is:

$$\mathtt{Opt}^-[u, j] = |\eta(u) - j| + \min_{\substack{0 \le j_1, \ldots, j_\ell \le j \\ \sum_{i=1}^{\ell} j_i = j}} \left\{ \sum_{i=1}^{\ell} \mathtt{Opt}[v_i, j_i] \right\}.$$

This corresponds to the optimal way of placing j pebbles on the subtrees rooted at the children of u.

As we have previously shown, the optimal assignment to $j_1, \ldots, j_\ell$ can be found in $O(\ell \cdot k)$ time by using dynamic programming. The value of the optimal solution is stored in $\mathtt{Opt}[r, k]$ and the solution itself can be reconstructed by proceeding backwards as shown in Section 3.1 for CON-SUM.

As we can compute the values $\mathtt{Opt}[u, j]$, for a fixed j and all $v \in V$, in $O(n \cdot k)$ time (the ℓ-values sum up to $n - 1$), the total time required by the algorithm is $O(n \cdot k^2)$.

The above discussion, immediately leads to the following:

Theorem 6. IND-SUM *on trees can be solved in* $O(n \cdot k^2)$ *time.*

4.3.2 Solving IND-NUM on trees.
The algorithm is similar to the one for IND-SUM: we root the tree G at an arbitrary vertex $r \in V(G)$ to obtain the rooted tree G_r, then we consider all the subtrees of G_r in a bottom-up fashion.

Let $\gamma(u)$ be 1 if at least one pebble is initially placed on the vertex $u \in V(G)$, and 0 otherwise. As before, when the subtree T_u, rooted at the vertex u, of G_r is examined, we consider an auxiliary problem where we want to place exactly $j \le k$ pebbles on the vertices of T_u such that their final positions induce an independent set of G (and each pebble is placed on a different vertex).

We will measure the cost of a solution for this auxiliary problem by examining the number of pebbles placed on each vertex of T_u. Removing pebbles from a vertex costs nothing, while placing a pebble on a vertex costs 1 if it comes from a different vertex. A way to visualize this auxiliary problem is to imagine the tree T_u where no pebbles have been placed, and a pool of j pebbles to be distributed on its vertices. If a vertex v is such that $\gamma(v) = 1$ then it can hold a single pebble for free, otherwise placing a pebble on v will increase the overall cost by 1.

We will denote by $\mathtt{Opt}[u, j]$ the cost of the optimal movement for this auxiliary problem. To solve the original problem we need to find a solution corresponding to $\mathtt{Opt}[r, k]$. We now show how to combine these auxiliary problems.

Let $\mathtt{Opt}^+[u, j]$ (resp., $\mathtt{Opt}^-[u, j]$) with $u \in V(G)$ and $0 \le j \le n$ be the value of an optimal solution to the auxiliary problem where exactly one pebble must

be placed on u (resp., no pebbles can be placed on u). Notice that $\mathtt{Opt}[u, j] = \min\{\mathtt{Opt}^+[u, j], \mathtt{Opt}^-[u, j]\}$. We will say that infeasible solutions have cost $+\infty$.

If u is a leaf of G_r then we have:

$$\mathtt{Opt}^+[u, j] = \begin{cases} +\infty & \text{if } j = 0 \text{ or } j \geq 2; \\ 1 - \gamma(u) & \text{if } j = 1. \end{cases} \qquad \mathtt{Opt}^-[u, j] = \begin{cases} 0 & \text{if } j = 0; \\ +\infty & \text{if } j \geq 1. \end{cases}$$

If u is not a leaf then, with a reasoning similar to that we did for IND-SUM we have:

$$\mathtt{Opt}^+[u, j] = \begin{cases} +\infty & \text{if } j = 0; \\ 1 - \gamma(u) + \min_{\substack{0 \leq j_1, \ldots, j_\ell < j \\ \sum_{i=1}^{\ell} j_i = j-1}} \left\{ \sum_{i=1}^{\ell} \mathtt{Opt}^-[v_i, j_i] \right\} & \text{if } j \geq 1, \end{cases}$$

and

$$\mathtt{Opt}^-[u, j] = \min_{\substack{0 \leq j_1, \ldots, j_\ell \leq j \\ \sum_{i=1}^{\ell} j_i = j}} \left\{ \sum_{i=1}^{\ell} \mathtt{Opt}[v_i, j_i] \right\},$$

where $v_1, \ldots, v_\ell$ are the children of u in G_r.

Again, the optimal assignment to $j_1, \ldots, j_\ell$ can be found in $O(\ell \cdot k)$ time by using dynamic programming. The value of the optimal solution is stored in $\mathtt{Opt}[r, k]$ and the solution itself can be reconstructed by proceeding backwards. As we can compute the values $\mathtt{Opt}[u, j]$, for a fixed j and all $v \in V$, in $O(n \cdot k)$ time (the ℓ-values sum up to $n - 1$), the total time required by the algorithm is $O(n \cdot k^2)$.

The above discussion immediately leads to the following:

Theorem 7. IND-NUM *on trees can be solved in $O(n \cdot k^2)$ time.*

4.3.3 Solving IND-MAX on paths.
In this section we concentrate on IND-MAX, and we devise an efficient algorithm for solving IND-MAX on the special case where G is a path. We start by proving the following:

Lemma 4. *Let $s_1, s_2, e_1, e_2 \in \mathbb{N}$ where $s_1 \leq s_2$ and $e_2 \leq e_1$, we have* $\max\{|s_1 - e_1|, |s_2 - e_2|\} \geq \max\{|s_1 - e_2|, |s_2 - e_1|\}$.

Proof. Suppose that $|s_1 - e_2| \geq |s_2 - e_1|$. If $s_1 \geq e_2$ then $|s_2 - e_2| \geq |s_1 - e_2|$, else if $s_1 < e_2$ then $|s_1 - e_1| = e_1 - s_1 \geq e_2 - s_1 = |s_1 - e_2|$.

In a similar manner suppose $|s_1 - e_2| \leq |s_2 - e_1|$. If $s_2 \geq e_1$ then $|s_2 - e_2| \geq |s_2 - e_1|$, else if $s_2 < e_1$ then $|s_1 - e_1| = e_1 - s_1 \geq e_1 - s_2 = |s_2 - e_1|$. $\qquad \square$

15

Let r be any endpoint of G, with a little abuse of notation we will identify the vertices of G by their distance from r: the vertex i with $0 \leq i < n$ will be the unique vertex u such that $d_G(r, u) = i$.

Let $p_1, \ldots, p_k$ be an ordering of the pebbles such that $\sigma(p_i) \leq \sigma(p_{i+1})$ for $i = 1, \ldots, k-1$. The following lemma shows that there exists an optimal solution where the ordering of pebbles is preserved, i.e., no edge of G is traversed by two pebbles going in opposite directions.[6]

Lemma 5. *If there exists a feasible solution μ for* IND-MAX *on paths then there also exists a solution μ' such that $\mu'(p_i) < \mu'(p_{i+1})$ for every $i = 1, \ldots, k-1$ and $c(\mu') \leq c(\mu)$.*

Proof. Let μ be a feasible solution. We will show that whenever there are two pebbles p_i and p_{i+1} such that $\mu(p_i) > \mu(p_{i+1})$, they can be swapped without increasing the cost of μ to obtain another feasible solution.[7] If needed, the procedure can be repeated until all the pebbles are placed in the right order.

In order to prove the above, consider a new solution μ' where $\mu'(p_i) = \mu(p_{i+1})$, $\mu'(p_{i+1}) = \mu(p_i)$, and $\mu'(p_j) = \mu(p_j)$ for every $j \notin \{i, i+1\}$. Clearly μ' is feasible (as we only swapped two pebbles) and requires at most the same maximum movement as μ, because:

$$\max \left\{ d_G(\sigma(p_i), \mu(p_i)), d_G(\sigma(p_{i+1}), \mu(p_{i+1})) \right\} \geq$$
$$\max \left\{ d_G(\sigma(p_i), \mu(p_{i+1})), d_G(\sigma(p_{i+1}), \mu(p_i)) \right\},$$

where every distance $d_G(u, v)$ between two vertices u and v can be rewritten as $|u - v|$ and Lemma 4 holds. $\square$

The algorithm takes as input an instance of IND-MAX plus a non-negative integer z and exploits the previous property to compute a solution of cost at most z, if it exists. The idea is simple: the pebbles are moved towards the endpoint r of the path as most as possible while preserving the feasibility of the solution and the constraint on the cost. The pseudocode is given below.

Lemma 6. *Algorithm 1 correctly returns a solution with maximum movement at most z, if such a solution exists, or reports infeasibility if there is no such solution.*

---

[6] This is not true if G is a tree: take the tree T with vertex set $\{1, 2, \ldots, 11\}$ and edge set $\{(i, i+1) : 1 \leq i \leq 4\} \cup \{(2, 6), (2, 7), (3, 8), (3, 9), (4, 10), (4, 11)\}$, place 2 pebbles on vertex 1 and one pebble on each vertex in $\{6, 7, 8, 9, 10, 11\}$. It is easy to see that any optimal solution has cost 3 and that at least one edge must be traversed in opposite directions.

[7] The case $\mu(p_i) = \mu(p_{i+1})$ is clearly impossible.

Algorithm 1: Algorithm for IND-MAX

Input : An instance $\langle G, P, \sigma \rangle$ of IND-MAX on paths, a non-negative integer z;
Output: A solution μ such that $c(\mu) \leq z$, iff such a solution exists.

$p_1, \ldots, p_k \leftarrow$ pebbles sorted by distance from r
$next \leftarrow 0$
for $j \leftarrow 1$ **to** k **do**
  $h \leftarrow \max\{next, \sigma(p_j) - z\}$

  if $h \geq n$ *or* $h > \sigma(p_j) + z$ **then**
  $\quad$ **return** No feasible solution exists

  $\mu(p_j) \leftarrow h$
  $next \leftarrow h + 2$

return μ

Proof. First notice that all solutions returned by the algorithm are feasible as no two pebbles can be placed on adjacent vertices and, by construction, no pebble travels a distance greater than z. This implies that if there is not a feasible solution, the algorithm correctly reports infeasibility.

Now we will prove that the algorithm correctly computes a solution when this exists. Let μ^* be a solution with maximum movement at most z that preserves the ordering of the pebbles (by Lemma 5 such a solution always exists). We will prove by induction that at the end of the j-th loop the first j pebbles are correctly placed (they are on an independent set), and that for the j-th pebble $\mu(p_j) \leq \mu^*(p_j)$ holds.

Base case: the first pebble is placed on the first reachable node from r, thus $\mu(p_1) \leq \mu^*(p_1)$ must hold.

Inductive step: suppose that the property is true for the first $j \geq 1$ pebbles, we will prove that it is also true also for the $(j+1)$-th pebble. If the pebble p_{j+1} is placed on vertex $\sigma(p_{j+1}) - z$ then the property holds, since this is the vertex with the smallest distance from r that is reachable by p_{j+1}. Otherwise $\mu(p_{j+1}) = \mu(p_j) + 2$, and we have $\mu^*(p_{j+1}) \geq \mu^*(p_j) + 2 \geq \mu(p_j) + 2 = \mu(p_{j+1})$. $\qquad\square$

We are ready to give the following:

Theorem 8. IND-MAX *on paths can be solved in* $O(n + k \log n)$ *time.*

Proof. From Lemma 6 it follows that IND-MAX on paths can be solved as follows: if $k > n$ then there is no solution, otherwise we can perform a binary search using Algorithm 1 to find the optimal value of k (or report infeasibility).

As far as the time complexity is concerned, each invocation of the algorithm requires time $O(k)$ and at most $O(\log n)$ invocations are required. For Algorithm 1 to work, vertices and pebbles must be sorted w.r.t. their distance from r, and this needs

to be done only once and requires $O(n + k)$ time. The whole procedure then takes $O(n + k \log n)$ time. □

5 Clique motion problems

In this section we prove that the problems CLIQUE-MAX, CLIQUE-SUM, and CLIQUE-NUM are NP-hard. Then, we give a tight approximation algorithm for CLIQUE-MAX that computes a solution that costs at most one more than the optimal solution. As of CLIQUE-SUM and CLIQUE-NUM, we show that the problems are not approximable within any factor smaller than $10\sqrt{5} - 21$, unless P = NP, and we devise two 2-approximation algorithms.

Actually, we will show that any approximation for CLIQUE-SUM or CLIQUE-NUM implies an approximation with the same ratio for the *minimum vertex cover* problem, which is known to be not approximable under $10\sqrt{5} - 21$, unless P = NP [9]. Moreover, if the unique game conjecture [13] is true, then both approximation algorithms are tight as the corresponding problems are not approximable within any constant factor better than 2 [14].

Finally, we show that for classes of graphs where we can find a *maximum-weight clique* in polynomial time, we can also solve CLIQUE-NUM in polynomial time.

5.1 Approximability of CLIQUE-MAX

We prove the following:

Theorem 9. CLIQUE-MAX *is* NP-*hard.*

Proof. We show a reduction from the problem of determining if there exists a *dominating clique* in a graph H, which is known to be NP-complete [12]. A dominating clique is a subset of vertices $C \subseteq V(H)$ such that C is both a clique and a dominating set for H.

We construct an instance of CLIQUE-MAX by setting $G = H$ and placing a pebble on each vertex of G. We claim that there exists a dominating clique in H if and only if the optimal solution for CLIQUE-MAX has measure c^* at most 1.

Suppose that there exists a dominating clique C in H. By definition, C is also a dominating set of G. We define μ so that every pebble initially placed on a vertex $u \notin C$ is moved to a vertex $v \in C$ such that $(u, v) \in E(G)$ (notice that such a vertex always exists). After their movement the pebbles are placed on a clique of G and each pebble has travelled a distance of at most 1.

Now suppose that there exists a solution μ for CLIQUE-MAX such that $c(\mu) \leq 1$. Clearly $\mu[P]$ is a clique, and it is also a dominating set, since for each vertex $u \notin \mu[P]$, there exists a vertex $v \in \mu[P]$ such that $(u, v) \in G$. □

On the positive side, we have:

Theorem 10. *It is possible to compute a solution $\tilde{\mu}$ for* CLIQUE-MAX *such that* $c(\tilde{\mu}) \leq c^* + 1$ *in polynomial time, where c^* is the measure of an optimal solution.*

Proof. Consider the following algorithm: for each vertex $u \in V(G)$ construct a solution μ_u that moves all the pebbles to u (i.e., set $\mu_u(p) = u$, $\forall p \in P$) and compute $c(\mu)$. Among the n possible solutions choose the one of minimum cost and call it $\tilde{\mu}$.

Now, let μ^* be an optimal solution. Recall that $c(\mu^*) = \max_{p \in P} \{d_G(\sigma(p), \mu^*(p))\}$. Let $C = \mu^*[P]$ be the clique where pebbles have been placed by μ^*, and notice that when the above algorithm considers a node $u \in C$ we have: $d_G(\sigma(p), \mu_u(p)) \leq d_G(\sigma(p), \mu^*(p)) + 1 \ \forall p \in P$, thus $c(\tilde{\mu}) \leq c(\mu_u) \leq c(\mu^*) + 1$. The claim follows. $\qquad\square$

5.2 Approximability of CLIQUE-NUM

We prove the following:

Theorem 11. CLIQUE-NUM *is 2-approximable and is not approximable within any constant factor smaller than $10\sqrt{5} - 21$, unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Let $\langle G, P, \sigma \rangle$ be an instance of CLIQUE-NUM and let $\varphi(u) = |\{p \in P : \sigma(p) = u\}|$ be the number of pebbles that are initially placed on vertex $u \in V$.

Let us assume that $\varphi(u) \leq 1$ for every $u \in V$, i.e., no two pebbles are placed on the same vertex. We will show later that this assumption is not restrictive.

Call H the graph induced by the vertices u with $\varphi(u) = 1$. Let $\bar{H}$ be the complement graph of H w.r.t. the edge set, that is the graph such that $V(\bar{H}) = V(H)$ and $E(\bar{H}) = \{(u, v) : u, v \in V(H) \land u \neq v \land (u, v) \notin E(H)\}$.

We will show that there exists a vertex cover $\mathcal{C}$ for $\bar{H}$ if and only if there exists a solution for the instance $\langle G, P, \sigma \rangle$ of CLIQUE-NUM of cost $|\mathcal{C}|$.

Let $\mathcal{C}$ be a vertex cover for $\bar{H}$, this implies that $Q = V(H) - \mathcal{C}$ is an independent set for $\bar{H}$, and therefore a clique for H and G. We construct a solution μ for CLIQUE-NUM by moving the $|\mathcal{C}|$ pebbles that are not yet placed on vertices in Q to one of such vertices.

Now let μ be a solution for the instance $\langle G, P, \sigma \rangle$ of CLIQUE-NUM and let $Q = \{u \in V(H) : \exists p \in P \text{ s.t. } \sigma(p) = \mu(p) = u\}$. Notice that the cost of μ is exactly $k - |Q| = |V(H) - Q|$. Clearly $Q \subseteq \mu[P]$ is a clique for G and H, therefore it is also an independent set for $\bar{H}$. This implies that $\mathcal{C} = V(H) - Q$ is a vertex cover for $\bar{H}$.

From the above it follows that the cost of an optimal solution is equal to the size of the minimum vertex cover for $\bar{H}$.

To approximate CLIQUE-NUM we construct the graph $\bar{H}$, we compute a 2-approximate minimum vertex cover $\tilde{\mathcal{C}}$, and then we reconstruct the solution μ as shown.

19

If the previous assumption is not met, i.e., there exists at least a vertex on which two or more pebbles are placed by σ, a slight modification to the instance is needed before we can apply the previous approach. We modify the graph G by replacing each vertex u such that $\varphi(u) > 1$ with a clique of size $\varphi(u)$. Each edge e incident to u is replaced by $\varphi(u)$ edges connecting every vertex of the clique to the other endpoint of e. Then, we modify the function σ so that the $\varphi(u)$ pebbles that were placed on u are assigned to each of the $\varphi(u)$ vertices of the corresponding clique. After the modifications, the cost of an optimal solution has not changed, moreover every solution for the modified instance can be easily reconverted to a solution for the original instance without increasing its cost. Hence, we have designed a 2-approximation algorithm for CLIQUE-NUM.

Concerning the inapproximability result, we prove it by contradiction: we show that if there exists an algorithm that approximates CLIQUE-NUM with a ratio better than $10\sqrt{5}-21$, then this would allow to approximate minimum vertex cover with the same approximation ratio. Indeed, let $\bar{H}$ be the instance of minimum vertex cover. Now, add an isolated vertex u and call G the complement of such a graph w.r.t. the edge set. Then, let $P = [|V(G)| - 1]$, and let σ be a function that places a single pebble on each vertex of G except u. We can now compute an approximate solution for the instance $\langle G, P, \sigma \rangle$ of CLIQUE-NUM and reconstruct a solution with the same cost for minimum vertex cover, as shown before.[8]   □

We close this section by proving that CLIQUE-NUM can be solved in polynomial time whenever we are able to solve the maximum-weight clique problem on a weighted variant of G. We remark that this is known to be the case of several classes of graphs, including perfect graphs. We refer the interested reader to [6] for a survey.

Theorem 12. *An exact solution to* CLIQUE-NUM *can be found in polynomial time on every class of graphs where a maximum-weight clique can be found in polynomial time.*

Proof. For every $v \in V$ let $\varphi(v)$ be the number of pebbles starting on vertex v, i.e., $\varphi(v) = |\{p \in P : \sigma(p) = v\}|$.

We consider a graph $H = (V, E)$ that is a copy of G where each vertex $v \in V$ has a weight $\varphi(v)$. Then, we compute a maximum-weight clique Q of H and construct a solution μ for CLIQUE-NUM by moving every pebble starting on a vertex in $V \setminus Q$ to an arbitrary vertex of Q.[9] It is now easy to see that this is an optimal solution, since for every other solution μ', we have:

$$c(\mu') = \sum_{v \notin \mu'[P]} \varphi(v) = k - \sum_{v \in \mu'[P]} \varphi(v) \geq k - \sum_{v \in Q} \varphi(v) = \sum_{v \notin \mu[P]} \varphi(v) = c(\mu).$$

---

[8] Notice that the additional vertex u is only instrumental to guarantee that G is connected.
[9] W.l.o.g. we assume that Q contains no vertices of weight 0.

$\square$

5.3   Approximability of Clique-Sum

We prove the following:

Theorem 13. Clique-Sum *is 2-approximable and is not approximable within any constant factor better than* $10\sqrt{5} - 21$, *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Let μ^* be an optimal solution to Clique-Sum. If μ^* moves all the pebbles (i.e., $\sigma(p) \neq \mu^*(p)$, $\forall p \in P$) then we can compute a 2-approximate solution $\tilde{\mu}$ by guessing a vertex $u \in \mu^*[P]$ and moving all the pebbles to u (i.e., setting $\tilde{\mu}(p) = u$, $\forall p \in P$). Indeed, we have:

$$\frac{c(\tilde{\mu})}{c(\mu^*)} = \frac{\sum_{p \in P} d_G(\sigma(p), \tilde{\mu}(p))}{\sum_{p \in P} d_G(\sigma(p), \mu^*(p))} \leq \frac{|P| + \sum_{p \in P} d_G(\sigma(p), \mu^*(p))}{\sum_{p \in P} d_G(\sigma(p), \mu^*(p))} \leq 2$$

where we used the fact that $d_G(\sigma(p), \tilde{\mu}(p)) \leq d_G(\sigma(p), \mu^*(p)) + 1$, and that $d_G(\sigma(p), \mu^*(p)) \geq 1$, for every pebble $p \in P$.

On the other hand, if there exists at least one pebble $p' \in P$ such that $\sigma(p') = \mu^*(p')$, then we guess its starting vertex $u = \sigma(p')$. We call P_0 the set of pebbles whose starting vertex is u, P_1 the set of pebbles whose starting vertex is adjacent to u, and P_2 the set of pebbles that are initially placed on a vertex at distance 2 or more from u. We then set $\tilde{\mu}(p) = u$ if $p \in P_0$ or $p \in P_2$. With a reasoning similar to the one of the previous case we can show that:

$$\sum_{p \in P_2} d_G(\sigma(p), \tilde{\mu}(p)) \leq |P_2| + \sum_{p \in P_2} d_G(\sigma(p), \mu^*(p)) \leq 2 \sum_{p \in P_2} d_G(\sigma(p), \mu^*(p)).$$

Concerning P_1, assume $P_1 \neq \emptyset$, and so we need to compute $\tilde{\mu}$ for the pebbles in P_1. To do that, consider the instance $\langle H, P_1, \sigma \rangle$ of Clique-Num where H is the subgraph of G induced by the vertices initially occupied by pebbles in $P_1 \cup \{p'\}$, and compute a 2-approximate solution μ' as shown in Theorem 11. Set $\tilde{\mu}(p) = \sigma(p)$ for every pebble $p \in P_1$ such that $\mu'(p) = \sigma(p)$, and set $\tilde{\mu}(p) = u$ for the remaining pebbles in P_1.

Clearly $\tilde{\mu}[P]$ is a clique for G as the vertices in $\mu'[P_1]$ are a clique for G, u is adjacent to every vertex in $\mu'[P_1]$, and $\tilde{\mu}[P] \subseteq \mu'[P_1] \cup \{u\}$.

Notice that the cost of moving the pebbles in P_1 w.r.t. μ^* is greater than or equal to the cost of the optimal solution for the instance $\langle H, P_1, \sigma \rangle$ of Clique-Num. Moreover, the cost of moving the pebbles in P_1 w.r.t. $\tilde{\mu}$ is equal to the cost of μ'. From the above it follows that:

$$\sum_{p \in P_1} d_G(\sigma(p), \tilde{\mu}(p)) \leq 2 \sum_{p \in P_1} d_G(\sigma(p), \mu^*(p)).$$

21

Therefore, the overall cost of this approximated solution is:

$$c(\tilde{\mu}) \le 2 \sum_{p \in P_1} d_G(\sigma(p), \mu^*(p)) + 2 \sum_{p \in P_2} d_G(\sigma(p), \mu^*(p)) \le 2\, c(\mu^*).$$

Concerning the inapproximability result, take a graph $H = (V, E)$ and construct the graph G by complementing H w.r.t. the edge set and adding an additional vertex v_0 adjacent to every other vertex. Let $P = [\|V(H)\|]$ and let σ be a function that places one pebble on each vertex of G except v_0. We will show that, given any solution for CLIQUE-SUM, it is possible to construct a vertex cover of H having the same cost, and vice versa. This implies that any approximation algorithm for CLIQUE-SUM converts into an approximation algorithm for minimum vertex cover with the same approximation ratio, therefore no approximation algorithm with an approximation ratio less than $10\sqrt{5} - 21$ can exist for CLIQUE-SUM [9], unless $\mathsf{P} = \mathsf{NP}$.

Let $\mathcal{C}$ be a vertex cover for H, then $V(H) - \mathcal{C}$ is an independent set for H and a clique for G. The solution that moves all the pebble of $\mathcal{C}$ to v_0 and leaves the others on their starting position is feasible and has a cost of $\mathcal{C}$.

Now let μ be a solution for the instance of CLIQUE-SUM. We modify μ in such a way that every pebble that moves at least by 1 is now moving to v_0. This modification cannot increase the cost of μ. Then, let $Q = \mu[P] \setminus \{v_0\}$, and let $\mathcal{C} = V(H) \setminus Q$. The cost of μ is $|V(H)| - |Q| = |\mathcal{C}|$, and $Q \subseteq \mu[P]$ is a clique for G. Therefore Q is also an independent set for H, and $\mathcal{C}$ is a vertex cover for H. $\qquad\square$

6 s-t-CUT motion problems

In this section we discuss (in)approximability results for s-t-CUT-MAX and s-t-CUT-SUM. Among the others, we provide an essentially tight approximation algorithm for s-t-CUT-MAX. Regarding s-t-CUT-NUM, establishing its tractability remains open, but we will show that approximating such a problem can be useful to approximate s-t-CUT-SUM, as well. We start by proving the following:

Theorem 14. *s-t-CUT-MAX and s-t-CUT-SUM are NP-hard even if G is a bipartite graph.*

Proof. We will show a reduction from the decision version of 3-SAT. Let $X = \{x_1, \ldots, x_\tau\}$ be the set of variables of the given formula f, and let m be the number of clauses. Start with an empty graph G and a set P of $\tau + 2m$ pebbles, then construct an instance for s-t-CUT-MAX and s-t-CUT-SUM as follows (see Figure 2 for an example):

- Add the two vertices s and t.
- For each variable $x_i \in X$ add a path of three vertices to G. Label the middle node u_i and the two endpoints x_i and $\bar{x}_i$, respectively. Place a pebble on u_i.
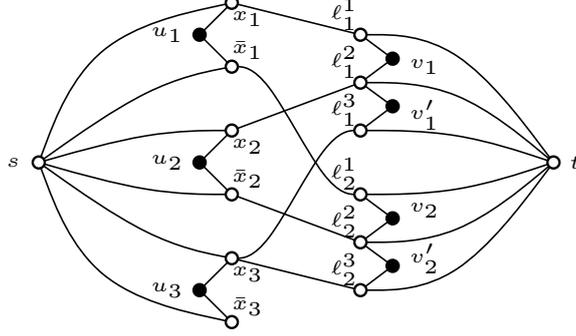
Fig. 2. Instance of s-t-CUT-MAX corresponding to the formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$. Pebbles are placed on black vertices. All the edges not incident to a black vertex represent paths of length h between their endpoints.

- For each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ create a path of five vertices labelled, from one endpoint to another, $\ell_1^j, v_j, \ell_2^j, v_j', \ell_3^j$. Place a pebble on v_j and one on v_j'.
- For each literal ℓ_j^i of f, let x_s be the corresponding variable. If ℓ_j^i is asserted add a new "long" path of length $h > k = t + 2m$ between the vertices labelled ℓ_j^i and x_s (so that $d_G(\ell_j^i, x_s) = h$). Otherwise add a new "long" path between the vertices labelled ℓ_j^i and $\bar{x}_s$.
- Connect s to every vertex x_i and to every vertex $\bar{x}_i$ with a new "long" path of length h.
- Connect t to every vertex ℓ_i^j with a new "long" path of length h.

Notice that G is bipartite as every cycle in G must pass trough an even number of long paths (paths of length h) and an even number of other edges.

We will show that there exists a truth assignment satisfying f if and only if an optimal solution for s-t-CUT-MAX (resp., s-t-CUT-SUM) has cost at most 1 (resp., k).

Suppose that there exists a truth assignment $\theta^* : X \to \{\text{true}, \text{false}\}$ that satisfies f. Then, for each $x_i \in X$ we move the pebble placed on u_i to x_i if $\theta^*(x_i) = \text{true}$, and to $\bar{x}_i$ if $\theta^*(x_i) = \text{false}$. Moreover, for each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ there are at most 2 literals whose truth values are false w.r.t. θ^*. We move one or two of the pebbles placed on v_j and v_j' to the nodes of G corresponding to the those literals.

Notice that in this way each pebble travels at most a distance of 1 (so the sum of the distances is at most k). To prove that the new positions of the pebbles induce an s-t-cut in G, consider any path $\pi(s, t)$ in G between s and t. We will show that $\pi(s, t)$ has been "blocked", i.e, there exists at least one pebble that has been moved on some vertex of $\pi(s, t)$.

The path $\pi(s, t)$ must contain (as subpath) at least one path of length h that connects a vertex $x \in \{x_i, \bar{x}_i\}$ for (some i) to a vertex $\ell \in \{\ell_j^1, \ell_j^2, \ell_j^3\}$ (for some j). If a

23

pebble has been placed on x then $\pi(s,t)$ is blocked. Otherwise the literal represented by ℓ must be false and, by construction, a pebble has been placed on ℓ.

Now suppose that an optimal solution for s-t-Cut-Max (resp., s-t-Cut-Sum) has cost at most 1 (resp., k). Notice that every pebble must have travelled a distance of at most 1. This is trivial for s-t-Cut-Max, while for s-t-Cut-Sum it suffices to note that placing a pebble on an internal vertex of the paths of length h blocks at most the same set of paths that are blocked by placing it on one of the endpoints. Moreover no pebble can traverse a whole path as $h > k$.

We define $\theta(x_i) = $ true if a pebble has been placed on x_i, and $\theta(x_i) = $ false otherwise. For each clause $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ there exists at least one vertex $\ell \in \{\ell_j^1, \ell_j^2, \ell_j^3\}$ such that no pebble has been moved to ℓ. Let x_i be the variable associated with ℓ_i, and let x be the vertex labelled x_i if $\ell = x_i$, or the vertex labelled $\bar{x}_i$ if $\ell = \bar{x}_i$. The pebble placed on u_i must have been moved to x, otherwise there would exist a path from s to t passing through x and ℓ, and therefore the clause is satisfied. $\qquad \square$

We now show that s-t-Cut-Max and s-t-Cut-Sum are actually very hard to approximate:

Theorem 15. *s-t-Cut-Max and s-t-Cut-Sum are not approximable within a factor of $n^{1-\epsilon}$ for every $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$. This also holds for bipartite graphs.*

Proof. As shown in the proof of Theorem 14, it is possible to construct instances of s-t-Cut-Max (and s-t-Cut-Sum) such that the optimal solution has measure at most 1 (resp., $k = \tau + 2m$) if and only if a boolean formula in conjunctive normal form with three literals per clause, τ variables, and m clauses is satisfiable. Moreover, any solution with cost z such that $\tau + 2m < z < h$ (recall that h is the length of the "long" paths) can be easily transformed into a solution of cost at most $\tau + 2m$ by moving every pebble p that has been placed on one long path to an appropriate endpoint (i.e., the one adjacent to $\sigma(p)$). This implies that if f is satisfiable, the measure z^* of an optimal solution is at most k for both the problems, while if f is not satisfiable, z^* is at least h.

Then, given a formula f, construct an instance for s-t-Cut-Max (and s-t-Cut-Sum) as in the proof of Theorem 14, and suppose that there exists a polynomial-time algorithm that approximates the optimal solution within a factor of $n^{1-\epsilon}$, for some positive $\epsilon \le 1$. Let $\tilde{c}$ be the measure of such an approximate solution. Observe that it is possible to upper bound the number of vertices n of the instance with the quantity $2 + 3\tau + 5m + 2\tau h + 6mh \le 11h(\tau + m) \le 11hk$.

If f is satisfiable, we have $\tilde{c} \le kn^{1-\epsilon}$, while if f is not satisfiable we have $\tilde{c} \ge h$. If $kn^{1-\epsilon} < h$ holds then it is possible to decide 3-Sat by running the approximation algorithm and looking the measure of the approximate solution. This can be guaranteed by choosing $h > (11k)^{2/\epsilon}$, as we have $h^\epsilon > 11k^2 \ge 11^{1-\epsilon}k^{2-\epsilon} = k(11k)^{1-\epsilon}$, which implies $h > k(11hk)^{1-\epsilon} \ge kn^{1-\epsilon}$. $\qquad \square$

By moving all the pebbles on a minimum s-t-cut of G we can show that the inapproximability result provided above is essentially tight for s-t-CUT-MAX:

Theorem 16. *s-t-CUT-MAX is d-approximable in polynomial time, where $d < n$ is the diameter of G, while s-t-CUT-SUM is $(k \cdot d)$-approximable in polynomial time.*

Proof. The approximation algorithm is as follows: if the initial position of the pebbles already makes s and t disconnected, then we are done. Otherwise, compute (in linear time) a minimum s-t-cut of G, say C. If $|C| > k$ this implies that both problems have no solution, otherwise move the pebbles on the vertices of C. The optimal measure must be at least 1, while each of the k pebbles travels a distance of at most d. □

We close this section proving a theorem which is useful in linking the approximability of s-t-CUT-NUM to that of s-t-CUT-SUM:

Theorem 17. *If there exists a ρ-approximation algorithm for s-t-CUT-NUM then there exists a $(\rho \cdot d)$-approximation algorithm for s-t-CUT-SUM, where $d < n$ is the diameter of G.*

Proof. Consider an instance of s-t-CUT-SUM an let c^* be the measure of an optimal solution. If c' is the measure of the optimal solution on s-t-CUT-NUM, then $c^* \geq c'$. If $c' = 0$ the claim is trivial, therefore we consider $c' > 0$. Then, a ρ-approximate solution for s-t-CUT-NUM is a $\rho \cdot d$-approximate solution for s-t-CUT-SUM, as each pebble travels a distance of at most d, and $\frac{\rho \cdot d \cdot c'}{c^*} \leq \frac{\rho \cdot d \cdot c'}{c'} = \rho \cdot d$. □

7 Conclusions

In this paper we have been concerned with the emerging field of (centralized) pebble motion problems. In particular, we have provided approximability and inapproximability results –most of which were tight– for several relevant variants studied in the literature. Among the issues we left open, the most prominent are those of establishing whether IND-MAX on trees can be solved in polynomial time, and of settling the computational complexity of s-t-CUT-NUM.

Motion planning of devices in a constrained environment deserves a further deep investigation in several respects. Here we have limited our attention to vertex-to-vertex motion on unweighted, undirected graphs, and with the objective of achieving very basic configurations, but it is easy to imagine more challenging scenarios. For instance, pebbles could be deployed on a 2-dimensional environment, or even on a terrain. In this case, the planning task could become substantially more difficult, because of the setting in the continuum. On the other hand, a simplifying yet very interesting scenario is that in which the given graph is a 2-dimensional grid. At an intermediate stage, an intriguing variant is that in which the underlying graph is weighted. We

also plan to look at other goal configurations, e.g., reaching a set of vertices inducing a 2-edge-connected subgraph, or connected subgraph with a bounded diameter, just to mention few.

References

1. Ahmadian, S., Friggstad, Z., Swamy, C.: Local-search based approximation algorithms for mobile facility location problems. *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, SIAM, 1607–1621, 2013.
2. Bilò, D., Disser, Y., Gualà, L., Mihal'ak, M., Proietti, G., Widmayer, P.: Polygon-constrained motion planning problems. *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSEN-SORS'13)*, LNCS 8243, Springer, 67–82, 2013.
3. Bilò, D., Gualà, L., Leucci, S., Proietti, G.: Exact and approximate algorithms for movement problems on (special classes of) graphs. *Proceedings of the 20th Colloquium on Structural Information and Communication Complexity (SIROCCO'13)*, LNCS 8179, Springer, 322–333, 2013.
4. Berman, P., Demaine, E.D., Zadimoghaddam, M.: $O(1)$-approximations for maximum movement problems. *Proocedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and of the the 15th International Workshop on Randomization and Computation (APPROX-RANDOM'11)*, LNCS 6845, Springer, 62–74, 2011.
5. Bollobás, B.: *Modern Graph Theory 184*, Springer, 1998.
6. Bomze, I.M., Budinich, M., Pelillo, M., Rossi, C.: Annealed replication: a new heuristic for the maximum clique problem. *Discrete Applied Mathematics* 121(1-3):27–49, 2002.
7. Demaine, E.D., Hajiaghayi, M.T., Mahini, H., Sayedi-Roshkhar, A.S., Gharan, S.O., Zadimoghaddam, M.: Minimizing movement. *ACM Transactions on Algorithms* 5(3), article 30, 2009.
8. Demaine, E.D., Hajiaghayi, M., Marx, D.: Minimizing movement: fixed-parameter tractability. *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, LNCS 5757, Springer, 718–729, 2009.
9. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 439–485, 2005.
10. Friggstad, Z., Salavatipour, M.R.: Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms* 7(3), article 28, 2011.
11. Hall, P.: On representatives of subsets. *Journal of the London Mathematical Society* 10(1):26–30, 1935.
12. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: *Domination in graphs: advanced topics 40*, Marcel Dekker, New York, 1998.
13. Khot, S.: On the power of unique 2-prover 1-round games. *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, ACM, 767–775, 2002.
14. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2-epsilon. *Journal of Computer and System Sciences* 74(3):335–349, 2008.
15. Prencipe, G., Santoro, N.: Distributed algorithms for autonomous mobile robots. *Proceedings of the 4th International Conference on Theoretical Computer Science (IFIP-TCS'06)*, IFIP 209, Springer, 47–62, 2006.