Reconfigurations in graphs and grids[*]

Gruia Călinescu[†] Adrian Dumitrescu[‡] János Pach[§]

May 21, 2007

Abstract

Let G be a connected graph, and let V and V' two n-element subsets of its vertex set $V(G)$. Imagine that we place a chip at each element of V and we want to move them into the positions of V' (V and V' may have common elements). A move is defined as shifting a chip from v_1 to v_2 ($v_1, v_2 \in V(G)$) on a path formed by edges of G so that no intermediate vertices are occupied. We give upper and lower bounds on the number of moves that are necessary, and analyze the computational complexity of this problem under various assumptions: labeled versus unlabeled chips, arbitrary graphs versus the case when the graph is the rectangular (infinite) planar grid, etc. We prove hardness and inapproximability results for several variants of the problem. We also give a linear-time algorithm which performs an optimal (minimum) number of moves for the unlabeled version in a tree, and a constant-ratio approximation algorithm for the unlabeled version in a graph. The graph algorithm uses the tree algorithm as a subroutine.

Keywords: Reconfiguration algorithms, approximation algorithms, local ratio method, proper function, NP-hardness.

1 Introduction

Consider a set system (set) of n pairwise disjoint objects in the Euclidean space that need to be brought from a given *start* (initial) configuration S into a desired *target* (goal) configuration T. In many cases, the problem admits the following abstraction: we have an underlying finite or infinite graph, the start configuration is represented by a set of n chips at n distinct start vertices and the target configuration by another set of n distinct target vertices. A vertex can be both a start and target position. The case when the chips are labeled or unlabeled give two different variants of the problem. In one move a chip can follow an arbitrary path in the graph and end up at another vertex, provided the path (including the end vertex) is free of other chips. The *motion planning* problem for such a system is that of computing a sequence of object motions (schedule) that achieves this task. If such a sequence of motions exists, we say that the problem is *feasible* and we say that it is *infeasible* otherwise. To avoid trivial questions, we always assume the graph is connected.

In certain applications, objects are indistinguishable, therefore the chips are unlabeled; for instance, a modular robotic system consists of a number of identical modules (robots), each of which having identical

[*]A preliminary version in *Proceedings of Latin American Theoretical Informatics Conference* (LATIN '06), vol. 3887 of LNCS, Springer, 2006, pp. 262–273.

[†]Dept. of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA; calinesc@iit.edu. Research partially supported by NSF grant CCF-0515088.

[‡]Dept. of Computer Science, University of Wisconsin–Milwaukee, WI 53201-0784, USA; ad@cs.uwm.edu. Research partially supported by NSF CAREER grant CCF-0444188.

[§]Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012-1185, USA; pach@cims.nyu.edu. Research partially supported by grants from NSF, NSA, OTKA, and from the US-Israeli Binational Research Foundation.

capabilities [10, 11]. In another application, the chips are indivisible packets (copies) of the same data that need to be moved from one site to another of a wide-area communication network without ever exceeding the capacities of the communication buffers at each site [5, 16].

In this variant with unlabeled chips, the problem is easier and always feasible; therefore one is interested in minimizing the number of moves. For the variant with labeled chips the problem may be infeasible: it is known for instance, that the 15-puzzle on a 4×4 grid — introduced by Sam Loyd in 1878 — has a solution if and only if the start permutation is an even permutation [15, 20] (see [3] for a recent approach).

Other reconfiguration rules (models) for systems of disks in the plane have been examined recently [1, 7, 8]; see also [9]. These models do not fall in the graph reconfiguration framework in this paper, because a disk may partially overlap several target positions. A model that fits in the graph reconfiguration framework has been analyzed in [10]: it deals with reconfiguration of modular systems acting in a grid-like environment, where moves must maintain connectivity of the whole system, and the motion rules are very local: a chip can only move to an adjacent position in one move. Denoting the configuration of the modules at time t by V_t, the system remains connected if for each $t = 0, 1, 2, \ldots$, the graph $G_t = (V_t, E_t)$ is connected, where E_t is the set of edges connecting pairs of cells in V_t that are side-adjacent.

The general form of the reconfiguration problem we consider is to find a reconfiguration sequence with a minimum number of moves. Depending on whether we refer to the graph or grid version, or to the labeled or unlabeled version, we call the problem U-GRAPH-RP, L-GRAPH-RP, U-GRID-RP or L-GRID-RP. In the grid version, the underlying graph is the infinite planar integer grid.

Consider for example the reconfiguration problem in the infinite grid with unlabeled (or labeled) chips (objects). The following simple algorithm does $2n$ moves for reconfiguration of n chips. In the first step (n moves), move in a suitable order all the chips away in the free grid space. In the second step (n moves), bring the chips ”back” to target positions. We will show that minimizing the number of moves is intractable in both (labeled and unlabeled) variants in the grid.

A move is a called *target move* if it moves a chip to a final target position. Otherwise it is called *non-target* move. Our lower bounds use the following argument: if no target chip coincides with a start chip (so each chip must move), a schedule with x non-target moves consists of at least $n + x$ moves.

It is worth noting that, in contrast with the variant with moves along free paths studied here, the variant of the reconfiguration problem (with unlabeled chips) in which we count as one move each edge traversed by a chip, can be solved exactly (optimally). This will be explained at the end of Section 2.

Previous related work: Most of the work done so far concerns labeled versions of the reconfiguration problem, and we give here only a very brief survey.

For the generalization of the 15-puzzle on an arbitrary graph (with $k = v-1$ labeled chips in a connected graph on v vertices), Wilson [21] gave an efficiently checkable characterization of the solvable instances of the problem. Kornhauser *et al.* have extended his result to any $k \leq v - 1$ and provided bounds on the number of moves for solving any solvable instance [16].

Ratner and Warmuth have shown that finding a solution with minimum number of moves for the $(N \times N)$-extension of the 15-puzzle is intractable [19], so the reconfiguration problem in graphs with labeled chips is NP-hard.

Auletta *et al.* gave a linear time algorithm for the *pebble motion on a tree* [5]. This problem is the labeled variant of the same reconfiguration problem we study here, however each move is along one edge only.

Papadimitriou *et al.* studied a problem of *motion planning on a graph* in which there is a mobile robot at one of the vertices s, that has to reach to a designated vertex t using the smallest number of moves, in the presence of obstacles (pebbles) at some of the other vertices [17]. Robot and obstacle moves are done along edges, and obstacles have no destination assigned and may end up in any vertex of the graph. The problem has been shown to be NP-complete even for planar graphs, and a ratio $O(\sqrt{n})$ polynomial time approximation algorithm was given in [17].

Dumitrescu *et al.* have addressed several basic questions in the analysis of modular metamorphic systems [11]. In particular the next two questions have been shown to be decidable: (*i*) whether a given set of motion rules maintains connectivity; (*ii*) whether a goal configuration is reachable from a given initial configuration (at specified locations) using a given set of motion rules. Other seemingly similar questions have been shown to be undecidable.

Our results are:

(1) The reconfiguration problem in graphs with unlabeled chips U-GRAPH-RP is NP-hard, and even APX-hard.

(2) The reconfiguration problem in graphs with labeled chips L-GRAPH-RP is APX-hard.

(3) For the infinite planar rectangular grid, both the labeled and unlabeled variants L-GRID-RP and U-GRID-RP are NP-hard.

(4) There exists a ratio 3 approximation algorithm for the unlabeled version in graphs U-GRAPH-RP. Thereby we also get a ratio 3 approximation algorithm for the unlabeled version U-GRID-RP in the (infinite) rectangular grid.

(5) We show that n moves are always enough (and sometimes necessary) for the reconfiguration of n unlabeled chips in graphs. For the case of trees, we present a linear time algorithm which performs an optimal (minimum) number of moves.

(6) We show that $7n/4$ moves are always enough, and $3n/2$ are sometimes necessary, for the reconfiguration of n labeled chips in the infinite planar rectangular grid (L-GRID-RP).

2 Chips in graphs

Let G be a connected graph, and let V and V' two n-element subsets of its vertex set $V(G)$. Imagine that we place a chip at each element of V and we want to move them into the positions of V' (V and V' may have common elements). A move is defined as shifting a chip from v_1 to v_2 ($v_1, v_2 \in V(G)$) along a path in G so that no intermediate vertices are occupied.

Theorem 1 *In G one can get from any n-element initial configuration V to any n-element final configuration V' using at most n moves, so that no chip moves twice. Moreover, for the case of a tree T with r vertices, there is a $O(r)$-time algorithm which performs the optimal (minimum) number of moves.*

Proof. It is sufficient to prove the theorem for trees. We argue by induction on the number of chips. Take the smallest tree T containing V and V', and consider an arbitrary leaf l of T. Assume first that the leaf l belongs to V: say $l = v$. If v also belongs to V', the result trivially follows by induction, so assume that this is not the case. Choose a path π in T, connecting v to an element v' of V' such that no internal point of π belongs to V'. Apply the induction hypothesis to $V \setminus \{v\}$ and $V' \setminus \{v'\}$ to obtain a sequence of at most $n-1$ moves, and add a final (unobstructed) move from v to v'.

The remaining case when the leaf l belongs to V' is symmetric: say $l = v'$; choose a path π in T, connecting v' to an element v of V such that no internal point of π belongs to V. Move first v to v' and append the sequence of at most $n-1$ moves obtained from the induction hypothesis applied to $V \setminus \{v\}$ and $V' \setminus \{v'\}$.

We further refine this algorithm so as to minimize the number of moves. We call a vertex that is both a start and a target position an *obstacle*. We have four types of vertices: free vertices, chip-only vertices, target-only vertices, and obstacles. Denote by c (resp. t) the number of chip-only (resp. target-only) vertices, and by o the number of obstacles. We have $c + o = o + t = n$. We call a tree *balanced* if it contains an equal number of chip-only and target-only vertices. Clearly, the initial tree T is balanced. If there exists an obstacle whose removal from T breaks T into balanced subtrees, we keep this obstacle fixed and proceed recursively (by induction) on the subtrees. If no obstacle removal breaks T into balanced subtrees, then all obstacles must move (each at least once), hence the number of moves necessary is at least $o + c = n$, and the algorithm in the first part of our proof can be used to obtain an optimal schedule.

We continue with the description of the linear time algorithm. We traverse the tree in postorder and compute for every vertex v four doubly-linked lists M_v, Q_v, A_v, and C_v. Let T_v denote the subtree rooted at v. We say that a sequence of moves *solves* T_v if, after executing the moves in this sequence, all the chips in T_v cover exactly every target of T_v (including obstacles). Note that by itself the definition of solving does not imply solving with a minimum number of moves.

We now discuss the four lists and give intuition on their use; a formal proof follows the description of the algorithm. A_v and C_v are lists of vertices inside T_v which require moves involving nodes outside T_v: for A_v those moves mean bringing a chip from outside T_v to a vertex of A_v, and for C_v those moves mean taking a chip from a vertex of C_v to a node outside T_v. Either A_v or C_v (possibly both) will be empty after node v is processed. A move is described by a pair of vertices: the place where we take hold of the chip and the place where we release it. M_v and Q_v are lists of moves inside T_v. Moves involving elements of A_v or C_v will use the edge from v to its parent. Moves not involving elements of $A_v \cup C_v$ will be done inside T_v and are all in M_v and Q_v. The goal is to obtain a sequence of moves solving T_v such that this sequence starts with the moves in M_v, followed by some moves on the edge from v to its parent (all these moves involve vertices from C_v or A_v), followed by the moves in Q_v. If v has no parent (i.e., v is the root of T), the list of moves on the edge from v to its parent is empty. The final reconfiguration sequence (output by the algorithm) is the sequence of moves in the list M_v followed by the moves in the list Q_v at the root of the tree.

We now describe how A_v, C_v, M_v, Q_v are initialized and computed. Two examples of the execution of the algorithm are shown in Figure 1. If v is a leaf, M_v and Q_v are empty, $A_v = < v >$ if v is a target-only vertex and empty otherwise, and $C_v = < v >$ if v is a chip-only vertex, and empty otherwise. We now show how the four lists M_v, Q_v, A_v, and C_v are obtained from the lists of the children of v. There are four cases, depending on whether v is free, target-only, chip-only, or an obstacle.

In all four cases, we first compute M_v, Q_v, A_v and C_v by concatenating the corresponding lists of the children of v. We also make use of a procedure *Cancel*, which works as follows: Iterate the following step until either A_v or C_v is empty: remove the first element from each list and add the corresponding pair at the end of M_v.

In the *first* case, v is a free vertex. After applying *Cancel*, we are done.

In the *second* case, v is a target-only vertex. We apply *Cancel*. Now, if C_v is empty, we put v at the end of A_v and we are done. If C_v is nonempty, we remove from it the last element q. Insert the move (q, v) at the beginning of Q_v, and we are done.

In the *third* case, v is a chip-only vertex. If A_v is empty, we put v at the beginning of C_v and we are done. If A_v is nonempty, we remove from it the first element q. We add the move (v, q) at the end of M_v.

Then we apply *Cancel* and we are done.

In the *fourth* and last case, v is an obstacle. (i) If A_v and C_v are both empty, we are done. (ii) If A_v is empty and C_v is nonempty, let q be the last chip from C_v. Remove q from C_v, insert the move (q, v) at the beginning of Q_v, add v at the beginning of C_v, and we are done. (iii) If A_v is nonempty, let p be the first target in A_v. Remove p from A_v and add the move (v, p) at the end of M_v. Apply *Cancel*. Now, if C_v is empty, add v at the end of A_v and we are done. If C_v is nonempty (note, A_v must be empty after *Cancel*), let q be the last chip from C_v. Remove q from C_v, insert the move (q, v) at the beginning of Q_v, and we are done.

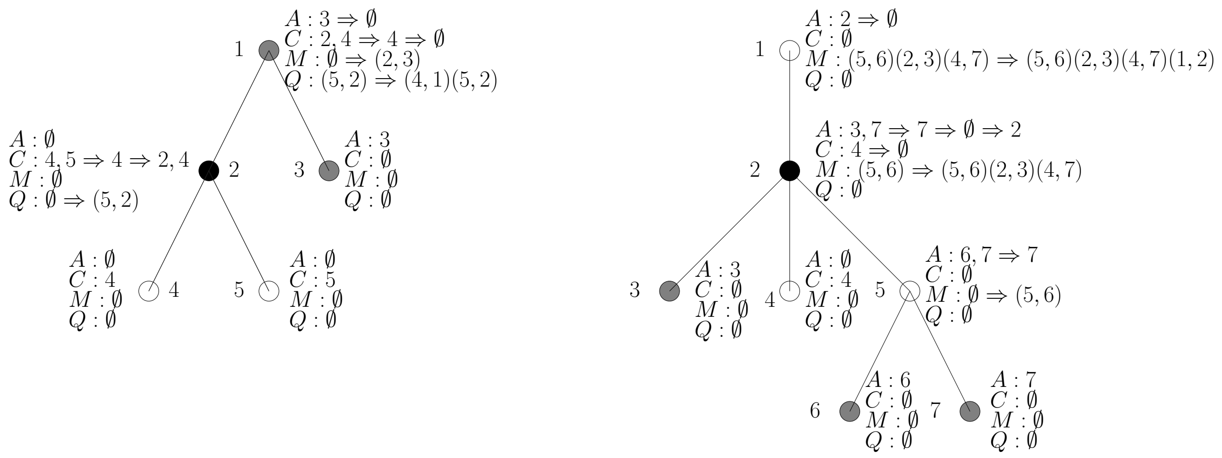

Figure 1: Two examples of execution of the linear time algorithm for reconfiguration in a tree. Chips are drawn as empty circles, obstacles as black circles, and targets as gray circles.

We need to prove this algorithm returns an optimum solution and then analyze its running time. First, we prove it returns a correct solution, and then we justify separately its optimality. For the correctness, we prove by induction on the size of T_v that the following three conditions hold:

- If T_v contains an equal number of start and target vertices, then $A_v = C_v = \emptyset$. The sequence of moves obtained by concatenating M_v and Q_v solves T_v.
- If T_v contains q more start vertices than target vertices, then $|C_v| = q$ and $A_v = \emptyset$. A sequence of moves obtained by concatenating M_v, any q moves taking chips from vertices of C_v in the order they appear in C_v and placing them on vertices outside T_v, and Q_v, solves T_v.
- If T_v contains q more target vertices than start vertices, then $|A_v| = q$ and $C_v = \emptyset$. A sequence of moves obtained by concatenating M_v, any q moves taking chips from vertices outside T_v and placing them on vertices of A_v in the order they appear in A_v, and Q_v, solves T_v.

The base case is obvious, and the inductive step follows directly from the description of the algorithm, thereby proving that the algorithm returns a correct solution. Regarding the optimality of the number of moves, notice that an obstacle v whose removal breaks T into balanced subtrees is not moved by the algorithm, and any other chip is moved only once: hence the algorithm returns an optimum solution.

The time spent at vertex v is proportional to one plus the number of children of v plus the number of moves added to M_v and Q_v. Summing over the vertices, we obtain that the time is proportional to the number of vertices in the tree plus the total number of moves, hence it is $O(n + r) = O(r)$, since $n \leq r$. $\square$

Remark. Theorem 1 implies that in the infinite rectangular grid, we can get from any starting position to any ending position of the same size n in at most n moves. It is perhaps interesting to compare this to the problem of sliding congruent unlabeled disks in the plane: here one can come up with "cage-like" constructions that require about $\frac{16n}{15}$ moves [7].

2.1 Hardness results

Theorem 2 *The unlabeled version in graphs* U-GRAPH-RP *is NP-complete. Moreover, assuming P $\neq$ NP, there is an absolute constant $\epsilon_1 > 0$ such that no polynomial-time algorithm has approximation guarantee at most $1 + \epsilon_1$. That is,* U-GRAPH-RP *is APX-hard.*

Proof. The decision version of U-GRAPH-RP is clearly in NP, so we only have to prove its NP-hardness. We reduce the *set cover* problem SC to U-GRAPH-RP. An instance of the set cover problem consists of a family $\mathcal{F}$ of subsets of a finite set U. The problem is to decide whether there is a set cover of size k for $\mathcal{F}$, i.e., a subset $\mathcal{F}' \subseteq \mathcal{F}$, with $|\mathcal{F}'| \leq k$, such that every element in U belongs to at least one member of $\mathcal{F}'$. SC is known to be NP-complete [12]. Consider an instance of SC represented by a bipartite graph $(B \cup C, E)$,

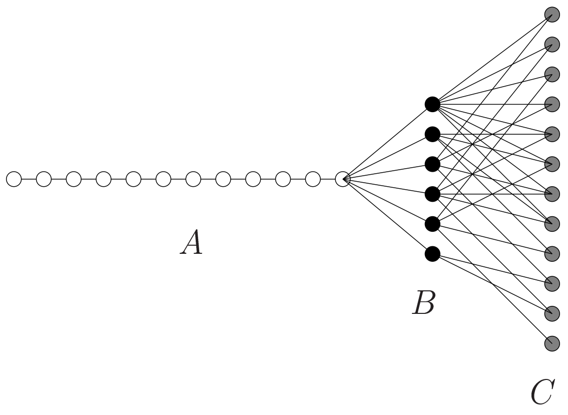

Figure 2: The "broom" graph G corresponding to a set cover instance with $|U| = 12$, and $|\mathcal{F}| = 6$. The vertices occupied only by chips are white, those occupied by both chips and targets are black, and those occupied only by targets are shaded. An optimal reconfiguration takes 15 moves (an optimal set cover has size 3: the third, fourth and fifth sets counting from the top).

where $U = C$, $\mathcal{F} = B$, and edges describe the membership relation. Construct the undirected graph G shown in Fig. 2, with $|A| = |C|$. The chips are $S = A \cup B$ and the targets are $T = B \cup C$. Clearly, G can be constructed in polynomial time. The reduction is complete once we establish the following claim.

Claim. There is a set cover consisting of at most q sets if and only if reconfiguration in G can be done using at most $|A| + q$ moves.

Proof of Claim: Let q be the size of an optimal set cover. The direct implication is clear: move one chip from each element of B in the optimal cover to a target in C; then move the elements of A to targets in C and to the emptied targets in B using the required number of moves. For the converse implication, assume that there is a reconfiguration sequence with fewer than $|A| + q$ moves. Since all elements of A must move, fewer than q elements of B move away from their original positions. By the definition of set cover, it means that some target in C will not be filled, a contradiction.

To prove the approximation hardness we use the same reduction, and the fact that 3-SC, the set cover problem in which the size of each set in $\mathcal{F}$ is bounded from above by 3 is APX-hard [2, 18]. Thus q, as in the paragraph above, is between $|A|/3$ and $|A|$. Approximating $|A| + q$ within $1 + \epsilon$ will approximate q within $1 + 4\epsilon$. □

A similar reduction can be made for the labeled version. The chips in A have targets in C, labeled as in Fig. 3 (here $|A| = |C| = m$). The obstacle chips in B coincide with their targets. Each vertex in B is

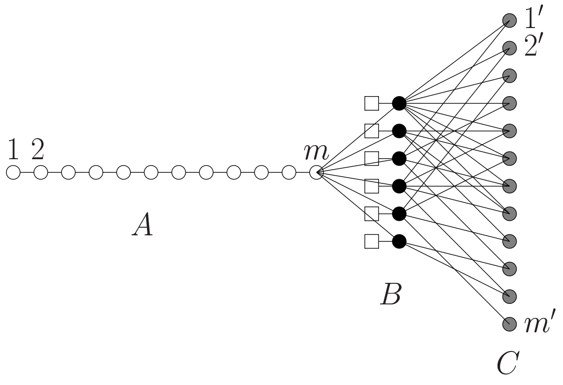

Figure 3: The graph constructed in the reduction for labeled chips. Free vertices are drawn as squares. An optimal reconfiguration takes 18 moves (an optimal set cover has size 3: the third, fourth and fifth sets counting from the top).

adjacent to a "twin" free vertex. The reduction follows from the next claim. Its proof is similar to that for previous claim, using the fact that a chip representing a selected set must move twice - once to to make way for the chips whose targets are on the vertices representing elements, and once to come back.

Claim. There is a set cover consisting of at most q sets if and only if reconfiguration in G can be done using at most $|A| + 2q$ moves.

We thus obtain:

Theorem 3 *The labeled version in graphs* L-GRAPH-RP *is APX-hard.*

2.2 Approximation algorithms

Theorem 4 *There exists a* 3*-approximation algorithm for* U-GRAPH-RP.

Proof. The algorithm is obtained by applying the local ratio method of Bar-Yehuda [6] to a graph H whose construction we describe below.

The vertex set of the input graph G is partitioned into four sets:

- $C = V \setminus V'$, the chip-only vertices
- $A = V' \setminus V$, the target-only vertices
- $B = V \cap V'$, the obstacles
- $F = V(G) \setminus (V \cup V')$, the free vertices

Then $V(H) = A \cup B \cup C$. Observe that $|A| = |C|$. For every pair of vertices u and v of H, we put in $E(H)$ the edge uv if $uv \in E(G)$ or there is a path in G from u to v having all the internal vertices from F.

A set of vertices (in $V(H)$) is called *balanced* if it contains an equal number of chip-only and target-only vertices. In H, we use the local ratio method to find a small set of edges Q such that every connected component D of $(V(H), Q)$ is balanced. We call this the U-STEINER problem. U-STEINER is a network design problem given by a 0-1 *proper function* [13, 14], problem for which both the primal-dual schema and the local ratio method were known to give a 2-approximation. The local-ratio 2-approximation algorithm is also implicit later in this section. We briefly recall here that for a ground set V, a 0-1 function $f : 2^V \to \{0, 1\}$ is proper (cf. [13, 14]) if it satisfies the following three conditions: (i) $f(V) = 0$, (ii) if A and B are disjoint, then $f(A \cup B) \leq \max(f(A), f(B))$, and (iii) f is symmetric, i.e. $f(S) = f(V - S)$ for all $S \subseteq V$.

In our case, the function, defined over all subsets of vertices, is one if the set is unbalanced and zero otherwise. It can be shown that this is a proper function, however this fact is not needed in the ratio 3 approximation algorithm, whose proof is self-contained. From the next claim, one can easily obtain a 4-approximation for U-GRAPH-RP, as shown after the proof of claim.

One more piece of notation: given a solution R for U-GRAPH-RP in G, consider the edges of G traversed by the moving chips during the reconfiguration process. These edges, together with their endpoints (including the free vertices through which chips pass through) form a number $k \geq 1$ of connected components. We then say that R has k connected components. Write $c = |C| = |A|$.

Claim 1 *Given a feasible solution R for* U-GRAPH-RP *in G with m moves and k connected components, there is a feasible solution Q for* U-STEINER *in H with at most $m + c - k$ edges. Conversely, given a feasible solution Q for* U-STEINER *in H with e edges, there is a feasible solution R for* U-GRAPH-RP *in G with at most $e - c + k$ moves, where k is the number of connected components of Q which intersect A (and C).*

Proof. For the first part, let S be the set of vertices of G involved in the moves of R, and let S_i, for $i = 1, 2, \ldots, k$, be the connected components of R. Then the number of moves inside S_i is at least $|S_i \cap C| + |S_i \cap B|$. Let $S'_i = S_i \cap V(H)$ and note that S'_i is also connected in H. In each S'_i pick a spanning tree T_i; the union of the trees T_i is the feasible solution Q for U-STEINER. In each T_i, the number of edges is $|V(T_i)| - 1 = |V(T_i) \cap C| + |V(T_i) \cap B| + |V(T_i) \cap A| - 1$. Summing over i gives the needed inequality.

For the second part, let T_i, for $i = 1, \ldots, k$, be one such connected component (w.l.o.g. a tree) with e_i edges. Then $|A \cap V(T_i)| = |C \cap V(T_i)|$ and using Theorem 1 one can move all the chips of $V(T_i)$, including those sitting on obstacles, to all the targets of $V(T_i)$ using $|V(T_i)| - |C \cap V(T_i)|$ moves: the chips from $(C \cup B) \cap V(T_i)$ move along the edges of T_i, passing if necessary through vertices of F (in G). Since $|V(T_i)| = |E(T_i)| + 1$, this second part of the claim follows by adding up over the components, since no move puts a chip on a vertex in F. (As a side remark, if Q is an optimal solution in H, each T_i is a tree intersecting A and C). □

Here is a short account for the ratio 4 approximation algorithm: By the first part of Claim 1 applied to an optimal solution for U-GRAPH-RP in G with m_{OPT} moves and k_{OPT} components, the number of edges e_{OPT} in an optimal solution for U-STEINER in H satisfies

$$e_{OPT} \leq m_{OPT} + c - k_{OPT} \leq m_{OPT} + c \leq 2m_{OPT}.$$

Therefore, by the second part of Claim 1, the number of moves m in the solution for U-GRAPH-RP in G returned by the algorithm satisfies (since $k_1 \leq c$, where k_1 is the number of components in the solution for U-STEINER in H which intersect A and C)

$$m \leq e - c + k_1 \leq e \leq 2e_{OPT} \leq 4m_{OPT}.$$

We now present the ratio 3 approximation algorithm. To this end, we have to enter the details of the local ratio method. The local ratio algorithm approximately solves U-STEINER instances with non-negative weight δ on the edges. The algorithm below, given in [6], is recursive. Each recursive call is given a set S of already selected edges and a non-negative edge weight function, and it returns a set of edges (in step 8). Given a set of edges $F \subseteq E(H)$, call a connected component X of $(V(H), F)$ *balanced* if $|V(X) \cap A| = |V(X) \cap C|$ and *unbalanced* otherwise. For the first call of the algorithm, $S = \emptyset$, and $\delta = 1$ on all edges. By feasible solution we mean feasible solution for the original U-STEINER instance.

1. The input parameters are the set of edges $S \subseteq E(H)$ and non-negative weight δ on $E(H)$.

2. If all the connected components of $(V(H), S)$ are balanced, return $\emptyset$.

3. Define weight function δ_1 on edges of $E(H)$ as follows: edges going between two unbalanced components of $(V(H), S)$ get weight 1, edges going between one unbalanced component of $(V(H), S)$ and one balanced component of $(V(H), S)$ get weight $1/2$, and all the other edges get weight 0.

4. Compute a positive real number α such that the weight function δ_2 on edges of H given by $\delta_2 = \delta - \alpha \cdot \delta_1$ is non-negative and for at least one edge e, we have $\delta_2(e) = 0 < \delta(e)$.

5. Let $M = \{e \mid \delta_2(e) = 0\}$.

6. Recursively solve the instance with parameters $S \cup M$ and weight δ_2 on $E(H)$, producing a set of edges L such that $S \cup M \cup L$ is a feasible solution.

7. Obtain minimal $M' \subseteq M$ such that $S \cup M' \cup L$ is a feasible solution.

8. Return $L \cup M'$.

We are guaranteed that $M \neq \emptyset$ and thus the algorithm terminates. For the approximation ratio, we need the following claim.

Claim 2 *During the execution of a recursive call of the algorithm, let k be the number of unbalanced components of $(V(H), S)$. Then $\delta_1(P) \geq k/2$ for any feasible solution P. The set of edges Q returned by the recursive call of the algorithm satisfies $\delta_1(Q) \leq k - q$, where q is the number of connected components of $(V(H), Q \cup S)$.*

Proof. Consider a feasible solution P. If an edge of P goes between two unbalanced components of $(V(H), S)$ we assign $1/2$ to each such component, and if it goes from one unbalanced component of $(V(H), S)$ to one balanced component of $(V(H), S)$, we assign $1/2$ to the unbalanced component. Each unbalanced component of $(V(H), S)$ must have at least one edge of P going to some other component, and thus it is assigned at least $1/2$. Therefore $\delta_1(P) \geq k/2$.

Consider now the edges Q selected (returned) by one recursive call of the algorithm and let Q_i, for $i = 1, \ldots, q$ be the connected components of $(V(H), Q \cup S)$. Fix one component Q_i. Inside Q_i, contract to a single vertex the vertices from the same component of $(V(H), S)$, obtaining $\bar{Q}_i$. The minimal property of Q as ensured by Step 7 of the algorithm ensures that $\bar{Q}_i$ is acyclic (and thus it is a tree) and every leaf of $\bar{Q}_i$ is an unbalanced component of S. Note that all the edges of Q_i with positive δ_1-weight are in $\bar{Q}_i$. Root $\bar{Q}_i$ at an arbitrary vertex v given by an unbalanced component of S.

For every $u \in (V(\bar{Q}_i) \setminus \{v\})$ given by an unbalanced component of S, charge to u either one or two edges, of total δ_1-weight 1, as follows: consider the path from u to v in $\bar{Q}_i$ and let u' be the next vertex on this path given by an unbalanced component of S. If the path has only one edge, charge this edge to u. If this path has more than one edge, charge to u the first and last edge of the path. It is easy to check that every edge of positive δ_1-weight of $\bar{Q}_i$ is charged at least once. Thus $\delta_1(Q_i) = \delta_1(\bar{Q}_i) \leq s_i - 1$, where s_i is the number of unbalanced component of S included in Q_i; here we used that v is not being charged.

Summing over i yields the second part of the claim. □

We continue with the proof of Theorem 4. First we note that, by using Claim 2 and induction, the local ratio algorithm ensures that its output LR satisfies, for any feasible solution P, the following:

$$\delta(LR) = \alpha\delta_1(LR) + \delta_2(LR) \leq \alpha 2\delta_1(P) + 2\delta_2(P) = 2\delta(P). \tag{1}$$

Let OPT be a solution with a minimum number of moves, $m(OPT)$ be the number of moves of OPT, and OP be the set of edges of the U-STEINER feasible solution obtained from OPT in Claim 1. Let LR be the set of edges selected by the local ratio approximation algorithm when applied to the U-STEINER instance, $k(LR)$ be the number of connected components of LR which intersect A, and $m(LR)$ be the number of moves of the solution obtained in Claim 1 from LR. The weight functions δ, δ_1, and δ_2 refer to the first call of the local ratio algorithm, as does the real α, which we note is at least 1. We have:

$$\begin{aligned}
m(LR) &\leq |LR| + k(LR) - c \quad \text{by Claim 1}\\
&= \delta(LR) + k(LR) - c\\
&= \alpha\delta_1(LR) + \delta_2(LR) + k(LR) - c\\
&\leq \alpha(2\delta_1(OP) - k(LR)) + \delta_2(LR) + k(LR) - c \quad \text{by Claim 2}\\
&\leq 2\alpha\delta_1(OP) - k(LR) + \delta_2(LR) + k(LR) - c \quad \text{since } \alpha \geq 1\\
&\leq 2\alpha\delta_1(OP) + 2\delta_2(OP) - c \quad \text{by Equation 1}\\
&\leq 2\delta(OP) - c\\
&= 2|OP| - c\\
&\leq 2(m(OPT) + c) - c \quad \text{by Claim 1}\\
&= 2m(OPT) + c\\
&\leq 3m(OPT), \quad \text{since } m(OPT) \geq c.
\end{aligned}$$

This concludes the proof of Theorem 4. □

Remark. In the graph version with unlabeled chips, if we count as one move every edge traversed by a chip, minimizing the number of moves can be solved in polynomial time, as described below. Construct a complete weighted bipartite graph $B = (V \cup V', E)$ with bipartition V: the vertices containing chips and V': the vertices containing targets (with obstacles in both sides of the bipartition). The weight of an edge in E is equal to the length of the shortest path in G connecting the endpoints of the edge. Now apply an algorithm for Minimum Weight Perfect Matching in B, and move accordingly: if the path a chip c_1 would take to reach its destination has another chip c_2 on it, have the two chips switch destinations and continue moving c_2. One can check that the number of moves does not exceed the weight of the perfect matching. On the other hand, the optimum solution must move chips to targets and cannot do better than the total length of the shortest paths in a minimum matching.

3 Chips in grids

In this section we analyze the reconfiguration problem with labeled, respectively unlabeled chips, in an infinite grid. However, a sufficiently large finite section of the grid containing all start and target positions, clearly suffices for this purpose.

3.1 Hardness results

Theorem 5 *The unlabeled version in the grid* U-GRID-RP *is NP-complete.*

Proof. The decision version of U-GRID-RP is clearly in NP, so we only have to prove its NP-hardness. We reduce the *Rectilinear Steiner Tree* problem R-STEINER to U-GRID-RP. An instance of R-STEINER consists of n points in the plane, and the problem is to decide whether there is a rectilinear Steiner tree (that is, a tree containing only horizontal and vertical edges) of length at most q which connects all points. For convenience the points can be chosen with integer coordinates. R-STEINER is known to be Strongly NP-complete [12] and thus we can assume all the coordinates are given in unary.

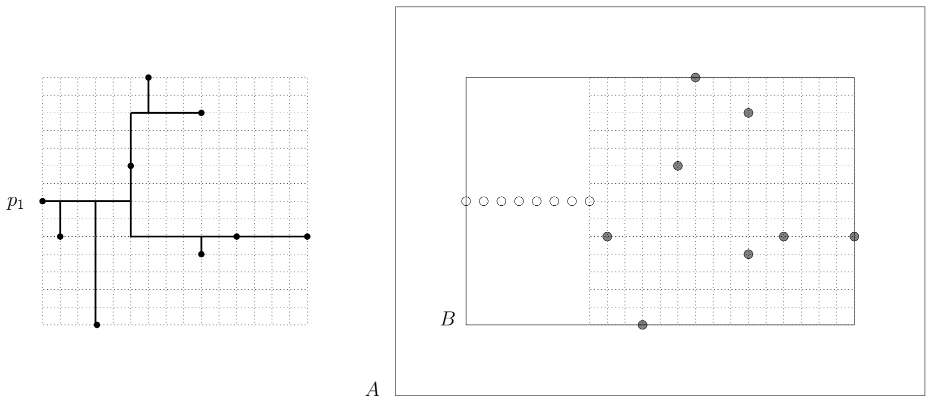

Figure 4: Unlabeled chips in the grid obtained from a Rectilinear Steiner Tree instance with $n = 9$ (sketch). Chip-only vertices are drawn as empty circles, and target-only vertices are drawn as filled (black) circles. Obstacles (not shown) are placed at each other integer point in the axis-aligned bounding rectangle.

Consider an instance $P = \{p_1, \ldots, p_n\}$ of R-STEINER with n points. Assume that the x-coordinate of a leftmost point $p_1 \in P$ is equal to zero. The instance of U-GRID-RP is illustrated in Fig. 4. It has $n-1$ chip-only vertices having the same y-coordinate with p_1 and x-coordinates $0, -1, -2, \ldots, -n+1$, and $n-1$ target-only vertices located at the other points $\{p_2, \ldots, p_n\}$ of the R-STEINER instance. Let B be a smallest axis-aligned bounding rectangle of all chip-only and target-only vertices, and Δ_x, Δ_y be the dimensions of B. Consider a sufficiently large axis-aligned rectangle A enclosing the chip-only and target-only vertices (thus A encloses B) with obstacles placed at each other integer point. The boundary of A is at distance $\Delta_x + \Delta_y$ from the boundary of B. This construction is done in time polynomial in $\Delta_x \times \Delta_y$, which is polynomial in the size of the R-STEINER instance since the coordinates are given in unary. The reduction is complete once we establish the following claim.

Claim. There is a rectilinear Steiner tree of length at most q if and only if reconfiguration can be done using at most q moves.

Proof of Claim: We start with the direct implication. Let T be a Steiner tree of length q connecting all target-only vertices and p_1. Pick a leaf of T which is an target-only vertex, and fill it with a chip (obstacle or chip-only) that is closest to it in T by moving it along the corresponding path in T. If the chip comes from an obstacle vertex, a new target-only vertex results in place of the leaf. The length of the resulting tree connecting all target-only vertices and p_1 is one less than the length of T. Continue by repeating this step until all targets have been filled (note that there is a leaf target-only vertex at each step); q moves are performed in total.

We continue with the converse implication. Denote by q the length of a minimum Steiner tree T connecting the points in P. Consider a valid reconfiguration sequence of moves $m_1, \ldots, m_t$, without loss of generality having a minimum number of moves. Each move m_i is an orthogonal path in the grid covering a set of elements $V(m_i)$ in rectangle B (since A is large enough, each move going outside rectangle B would result in a non-optimal reconfiguration sequence). In addition, we can assume that all chips pass through p_1. Note that $V = V(m_1) \cup \ldots \cup V(m_t)$ is the vertex set of a connected grid graph G that includes all chip-only elements, all target-only elements, and other obstacles.

The number of edges of G is at least $q + (n-2)$, as in addition to the Steiner tree $n-2$ edges appear to the left of p_1; thus $|V| \geq q + (n-2) + 1$. Note that $V \setminus \{p_2, \ldots, p_n\}$ consists of chips or obstacles. Each of these chips or obstacles must participate at least once in a move, and their number is at least $(q+1) + (n-2) - (n-1) = q$, as required.

This also concludes the proof of Theorem 5. □

Remark. Conform with the results of [4], there exists a polynomial-time approximation scheme for Rectilinear Steiner Tree. Thus our reduction does not give an inapproximability result for U-GRID-RP, and we leave open the question of whether U-GRID-RP admits a polynomial-time approximation scheme or not.

Theorem 6 *The labeled version in the grid* L-GRID-RP *is NP-complete.*

Proof. The argument, similar to that in the proof of Theorem 5, still uses a reduction from R-STEINER. Consider an instance $P = \{p_1, \ldots, p_n\}$ of R-STEINER with n grid points and let p_n be a rightmost point in P. The instance of L-GRID-RP is illustrated in Fig. 5. It has n labeled chips $s_1, \ldots, s_n$ located at the

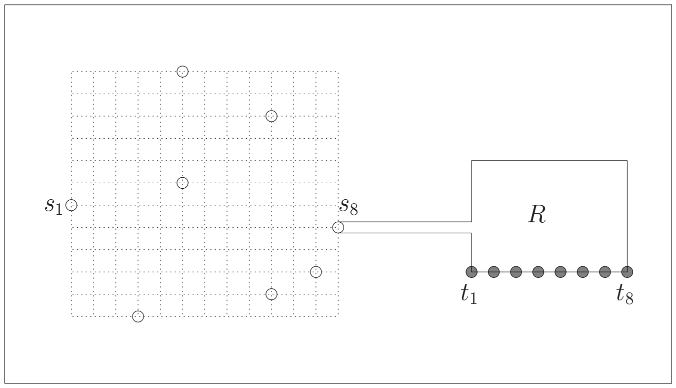

Figure 5: Labeled chips in the grid obtained from a Rectilinear Steiner Tree instance with $n = 8$ (sketch). Chip-only vertices are drawn as empty circles, and target-only vertices are drawn as filled (black) circles. Obstacles (not shown) are placed at each other integer point in the axis-aligned bounding rectangle. The corridor and most of the target area R (except its bottom row) are free.

points in P (s_i at p_i). Consider a sufficiently large axis-aligned rectangle enclosing the n chips mentioned above and filled with obstacles placed at each other integer point, except the following: make a thin (width is one) long corridor connecting the rightmost chip s_n to a large rectangular "target area" R located right of all the n chips mentioned above; both the corridor and all rows of R except its bottom row are free. The bottom row contains all n targets $t_1, \ldots, t_n$; here t_i denotes the target of s_i. R has $\Delta_x + \Delta_y + 2$ rows and n columns, where Δ_x, Δ_y are as before.

Informally, the obstacles which form a minimum rectilinear Steiner tree of the n points must go out in the target area and then come back in to their original positions. More precisely, the reduction is complete once we establish the following claim (analogous to the claim in the proof of Theorem 5).

Claim. There is a rectilinear Steiner tree of length at most q if and only if reconfiguration can be done using at most $2q + 2 - n$ moves.

This concludes the proof of Theorem 6. □

3.2 Labeled chips: upper and lower bounds on the number of moves

Theorem 7 *For the reconfiguration of n labeled chips in the infinite planar rectangular grid (*L-GRID-RP*), $7n/4$ moves are always enough, while $3n/2$ moves are sometimes necessary.*

Proof. The lower bound is trivial (however it does not appear to be trivial to improve on it!): take a pair of chips labeled 1 and 2, say next to each other, and have the target positions switch them; that is $t_1 = s_2$ and $t_2 = s_1$. Clearly three moves are needed to rearrange this group of two, and by repeating it (with different labels), one gets a pair of configurations which require $3n/2$ moves.

We now describe a reconfiguration algorithm which executes no more than $7n/4$ moves (as mentioned in the introduction, the problem can be solved trivially in $2n$ moves).

Let S and T be the start and target configurations. Consider the directed graph G with n edges (loops allowed) given by $S \rightarrow T$. Vertices are grid points of $S \cup T$ (the number of vertices is between n and $2n$). Each edge originates at a start chip and ends at some (free or occupied) target cell. Note that each in-degree and out-degree is at most one, so G can be partitioned into a collection of disjoint paths and cycles (and loops).

Consider the rows of S numbered from top to bottom: $1, 2, \ldots, r$. Let D (resp. E) be the set of elements in the odd (resp. even) rows; we can assume without loss of generality that $|D| \leq |E|$, thus $|D| \leq n/2$. Let A be the set of elements of E whose target lie in E, and let B (resp. C) be the set of elements of E whose target lie in rows of D congruent to 1 (resp. to 3) modulo 4. Write $a = |A|$, $b = |B|$, $c = |C|$. We can assume without loss of generality that $c \leq b$.

1. Move (far) away the elements of D (row by row, and for each row, move elements one by one, say from left to right) to form a set of corridors.

2. Move away the elements of C (elements of the even rows are adjacent to corridors, therefore any subset of chips of an even row can be moved away).

3. Select and move away an element from each cycle of the directed graph G remaining among the elements of A (not from the loops).

4. Fill (say, from left to right) the odd rows congruent to 1 modulo 4 with the elements of B and elements far away as follows: note that each even row is adjacent to an odd row congruent to 3 modulo 4; take out an element of B from the even row through the empty corridor congruent to 3 modulo 4, and then back in the target odd row congruent to 1 modulo 4.

5. Fill the even rows using the adjacent empty corridors (congruent to 3 modulo 4), with elements from A and elements far away. The elements of A move directly to their destination and such a move is possible as long as some elements of A still need to move, since we moved away one element from each cycle of the directed graph G contained in A.

6. Fill (say, from left to right) the odd rows congruent to 3 modulo 4 (the corridors) with elements far away.

The number of non-target moves is at most

$$n - (a + b + c) + \frac{a}{2} + c \leq n - \frac{a}{2} - b \leq \frac{3n}{4},$$

since $a + 2b \geq a + b + c \geq n/2$. Therefore the total number of moves is not more than $n + 3n/4 = 7n/4$. □

Remark. The above lower bound clearly holds even in the stronger *lifting model*, when chips can be lifted and placed back in the plane (see [7, 8] for related aspects of disk reconfiguration problems).

Acknowledgment. The authors thank Sergey Bereg and Marius Zimand for several conversations on the topic.

References

[1] M. Abellanas, S. Bereg, F. Hurtado, A. G. Olaverri, D. Rappaport, and J. Tejel, Moving coins, *Computational Geometry: Theory and Applications*, **34** (2006), 35–48.

[2] P. Alimonti and V. Kann, Hardness of approximating problems on cubic graphs, *Proceedings of the 3rd Italian Conference on Algorithms and Complexity*, LNCS 1203, Springer-Verlag (1997), 288–298.

[3] A. Archer, A modern treatment of the 15 puzzle, *American Mathematical Monthly*, **106** (1999), 793–799.

[4] S. Arora, Nearly linear time approximation schemes for Euclidean TSP and other geometric problems, *Journal of the ACM*, **45** (1998), 1–30.

[5] V. Auletta, A. Monti, M. Parente, and P. Persiano, A linear-time algorithm for the feasibility of pebble motion in trees, *Algorithmica*, **23** (1999), 223–245.

[6] R. Bar-Yehuda, One for the price of two: a unified approach for approximating covering problems, *Algorithmica*, **27** (2000), 131–144.

[7] S. Bereg, A. Dumitrescu, and J. Pach, Sliding disks in the plane, *International Journal of Computational Geometry & Applications*, to appear.

[8] S. Bereg and A. Dumitrescu, The lifting model for reconfiguration, *Discrete & Computational Geometry*, **35** (2006), 653–669.

[9] A. Dumitrescu, Motion planning and reconfiguration for systems of multiple objects; in *Mobile Robots: Perception & Navigation*, Sascha Kolski (editor), Advanced Robotic Systems, 2007, pp. 523–542.

[10] A. Dumitrescu and J. Pach, Pushing squares around, *Graphs and Combinatorics*, **22** (2006), 37–50.

[11] A. Dumitrescu, I. Suzuki and M. Yamashita, Motion planning for metamorphic systems: feasibility, decidability and distributed reconfiguration, *IEEE Transactions on Robotics and Automation*, **20** (2004), 409–418.

[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.*, W. H. Freeman and Company, 1979.

[13] M. X. Goemans and D. P. Williamson, A general approximation technique for constrained forest problems, *SIAM Journal on Computing*, **24** (1995), 296–317.

[14] M. X. Goemans and D. P. Williamson, The primal-dual method for approximation algorithms and its application to network design problems, in *Approximation Algorithms for NP-Hard Problems*, edited by D. S. Hochbaum, PWS Publishing Co., 1995.

[15] W. W. Johnson, Notes on the 15 puzzle. I., *American Journal of Mathematics*, **2** (1879), 397–399.

[16] D. Kornhauser, G. Miller, and P. Spirakis, Coordinating pebble motion on graphs, the diameter of permutation groups, and applications, *Proceedings of the 25-th Symposium on Foundations of Computer Science*, (FOCS '84), 241–250.

[17] C. Papadimitriou, P. Raghavan, M. Sudan, and H. Tamaki, Motion planning on a graph, *Proceedings of the 35-th Symposium on Foundations of Computer Science*, (FOCS '94), 511–520.

[18] C. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *Journal of Computer and System Sciences*, **43** (1991), 425–440.

[19] D. Ratner and M. Warmuth, Finding a shortest solution for the $(N \times N)$-extension of the 15-puzzle is intractable, *Journal of Symbolic Computation*, **10** (1990), 111–137.

[20] W. E. Story, Notes on the 15 puzzle. II., *American Journal of Mathematics*, **2** (1879), 399–404.

[21] R. M. Wilson, Graph puzzles, homotopy, and the alternating group, *Journal of Combinatorial Theory, Series B*, **16** (1974), 86–96.