Reconfigurations in graphs and grids

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

July 2, 2005

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 ($v_1, v_2 \in V(G)$) along 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 provide various 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.

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 goal (target) 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 start vertices and the target configuration by another set of n 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 capabilities [8, 9]. In this variant with unlabeled chips, the problem is easier and always feasible; therefore we are interested in minimizing the number of moves. For the variant with labeled chips it may be the case that the problem is infeasible: it is known for instance that the 15-puzzle on a 4×4 grid — introduced by

[*]Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA; calinesc@cs.iit.edu

[†]Computer Science, University of Wisconsin–Milwaukee, 3200 N. Cramer Street, Milwaukee, WI 53211, USA; ad@cs.uwm.edu

[‡]Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012-1185, USA; pach@cims.nyu.edu.
Supported by NSF grant CCR-00-98246, by an NSA grant, a PSC-CUNY Research Award, and grant OTKA-T-032-452 from the Hungarian Science Foundation.

Sam Loyd in 1878 — has a solution if and only if the start permutation is an even permutation [12, 16] (see [2] for a recent approach).

Other reconfiguration rules (models) for systems of disks in the plane have been examined recently: [1, 7, 6]. 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 [8]: 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.

Consider for example the reconfiguration problem in the infinite grid with unlabeled (or labeled) chips. The following simple algorithm does $2n$ moves for reconfiguration of n objects (chips). In the first step (n moves), move in a suitable order all the chips away anywhere 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. A move is a *target move* if it moves a chip to a final target position. Otherwise it is a *non-target* move. Our lower bounds use the 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.

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 $k = n - 1$ labeled chips in a graph on n vertices), Wilson gave an efficiently checkable characterization of the solvable instances of the problem, depending on whether the graph is bipartite or not [17]. Kornhauser *et al.* have extended his result to any $k \leq n - 1$ and provided bounds on the number of moves for solving any solvable instance [13].

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 [15], 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* [4]. 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 [14]. 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 [14].

Dumitrescu *et al.* have addressed several basic questions in the analysis of modular metamorphic systems [9]. 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 is NP-hard, and even APX-hard.

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

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

(4) There exists a ratio 3 approximation algorithm for the unlabeled version in graphs. Thereby we get a ratio 3 approximation algorithm for the unlabeled version in the (infinite) rectangular grid.

(5) We show that n moves are always enough (and sometimes necessary) for the reconfiguration of 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 labeled chips in the infinite planar rectangular grid.

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 edges of 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 $O(n + r)$-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 P in T, connecting v to an element v' of V' such that no internal point of P belongs to V'. Apply the induction hypothesis to $V - v$ and $V' - v'$ 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 P in T, connecting v' to an element v of V such that no internal point of P 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 $V - v$ and $V' - v'$.

We further refine this algorithm so as to minimize the number of moves. We call a vertex that is both a start and target position an *obstacle*. We have four types of vertices: free vertices, chips, targets and obstacles. Denote by c (resp. t) the number of chips (resp. targets), 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 chips and targets. 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.

To obtain a 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 be the subtree rooted at v. M_v and Q_v are lists of moves (pairs of vertices: the place where we take hold of the chip and the place where we release it) inside T_v, meaning roughly (details below) that an optimum solution for T can be obtained by first doing the moves in M_v, followed by moves on the edge from v to its parent (if v has no parent, this list of moves is empty), followed by the moves in Q_v. Let A_v and C_v be lists of nodes satisfying the following conditions:

- A_v (the list of remaining targets) or C_v (the list of remaining chips) is empty.
- If A_v and C_v are both empty, there is an optimum solution which makes the moves in M_v, followed by the moves in Q_v, followed by an optimum solution in $T \setminus T_v$.
- If A_v is empty, there is an optimum solution for the whole tree, which first applies the moves from M_v, then makes some other moves among which the chips from C_v are moved in the order of C_v on the edge from v to its parent, followed by the moves of Q_v.
- If C_v is empty, there is an optimum solution for the whole tree, which first applies the moves from M_v, then makes some other moves among which the chips from outside T_v are moved to the targets in A_v, the targets being used in the order of A_v, followed by the moves of Q_v.

If v is a leaf, M_v and Q_v are empty, $A_v =< v >$ if v is a target or empty otherwise, and $C_v =< v >$ if v is a chip or empty otherwise. We show how the four list M_v, Q_v, A_v, and C_v are obtained from the list of the children of v. There are four cases, depending if v is a chip, target, obstacle, or free.

In all for cases, we first compute M_v, Q_v, A_v and C_v by concatenating the lists of the children of v. We also make use of a procedure *Cancel* which works as follows. Iterate the following 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. 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 not empty, we remove from C_v q, its last element. Insert the move $q - v$ at the beginning of Q_v, and we are done.

In the third case, v is a chip. If A_v is empty, we put v at the beginning of C_v and we are done. With A_v nonempty, we remove from it q, the first element. 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. If A_v and C_v are both empty, we are done. If only A_v is empty, then 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. If A_v is not empty, 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 only A_v is empty, 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.

One can 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: thus the algorithm returns an optimum solution. The time spent at vertex v is proportional to 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. □

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 at least about $\frac{16n}{15}$ moves [6].

2.1 Hardness Results

Theorem 2 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* $1 + \epsilon_1$.

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 [10].

Consider an instance of SC represented by a bipartite graph $(B \cup C, E)$, where $U = C$, $\mathcal{F} = B$, and edges describe the membership relation. Construct the undirected graph G shown in Fig. 1, 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.

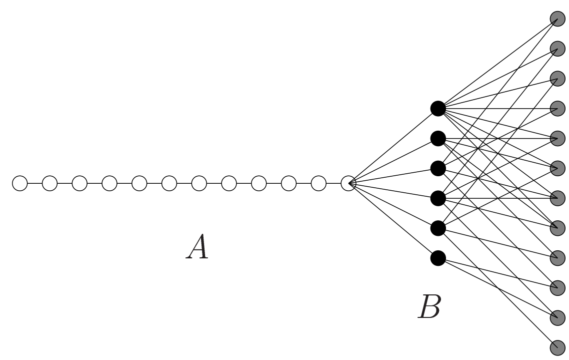

Figure 1: The "broom" graph G corresponding to a set cover instance with $|U| = 12$, and $|\mathcal{F}| = 6$. The vertices occupied by chips are white, those occupied by both chips and targets are black, and those occupied by targets are shaded. An optimal reconfiguration takes 15 moves (an optimal set cover has size 3).

The reduction is complete once we establish the following claim (proof omitted).

Claim 1 *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.*

To prove the approximation hardness we use the same reduction, and the fact that b-SC, the set cover problem in which the size of each set in $\mathcal{F}$ is bounded from above by a constant b is APX-hard [**?**]. ADRIAN SAYS: What reference should I put here? We omit the details. ⟵

Remark. A similar reduction can be made for the labeled version. The chips in A have targets in C, labeled as in Fig. 2 (here $|A| = |C| = m$).

The obstacle chips in B coincide with their targets. Each vertex in B is adjacent to a "twin" free vertex. The reduction follows from the next claim (proof omitted).

Claim 2 *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.*

□

2.2 Approximation algorithms

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

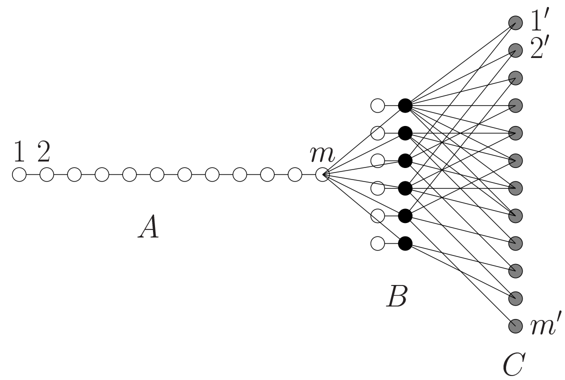

Figure 2: The graph constructed in the reduction for labeled chips. An optimal reconfiguration takes 18 moves (an optimal set cover has size 3).

Proof. The algorithm is obtained by applying the local ratio method of Bar-Yehuda [5] 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 chips
- $A = V' \setminus V$, the targets
- $B = V \cap V'$, the obstacles
- $F = V(G) \setminus (V \cup V')$, the free vertices

Then $V(H) = A \cup B \cup C$. 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.

In H, we use the standard local ratio method to find a set of edges Q such that every connected component D of $(V(H), Q)$ has an equal number of chips and targets. We call this the U-STEINER problem. U-STEINER is a network design problem given by a 0-1 proper function [11] and both the primal-dual schema and the local ratio method give a 2-approximation. The function, defined over all sets of vertices, is one if the set is unbalanced and zero otherwise. Recall that a set is *balanced* if it contains an equal number of chips and targets. 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 Q 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 say that Q has k connected components. Write $c = |C| = |A|$.

Claim 3 *Given a feasible solution Q for* U-GRAPH-RP *in G with m moves and k connected components, there is a feasible solution 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 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 Q, and let S_i, for $i = 1, 2, \ldots, k$, be the connected components of S. 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 each S'_i pick a tree T_i; the union of the trees T_i is the feasible solution of U-STEINER. In each T_i, the number of edges is $|V(T_i)| - 1 = |V(T_i) \cap C| + |V(T_i) \cap B| + |V(T_i) + A| - 1$. Summing over i gives the needed equality.

For the second part, let T_i, for $i = 1, \ldots, k$, be one such connected component 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 $|V(T_i)| - |C \cap T_i|$ 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 of F. (As a side remark, if Q is an optimal solution in H, T_i is a tree and all components intersect A and C). □

Here is a short account for the ratio 4 approximation algorithm: By the first part of Claim 3 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 3, 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)

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

We now present the ratio 3 approximation algorithm. We have to enter the details of the local ratio method to get this ratio (instead of 4). The local ratio approximately solves U-STEINER instances with non-negative weight δ on the edges. The algorithm below, given in [5], is recursive. Each recursive call is given a set S of already selected edges. 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 of 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 $E(H) \backslash S$ 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. Select 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 there is at least one edge e with $\delta_2(e) = 0 < \delta(e)$.
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 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 4 *During the execution 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 selected by 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 assigned $1/2$ 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 components of $(V(H), S)$ must have one edge of P going to some other component, and thus it is assigned at least $1/2$. Thus $\delta_1(P) \geq k/2$.

Consider now the edges Q selected by 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 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 3. First we note that, using induction, the local ratio algorithm ensured that its output LR satisfies, for any feasible solution P, the following:

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

Let OPT be a solution with minimum number of moves, $m(OPT)$ be the number of moves of OPT, OP be the set of edges of the U-Steiner feasible solution obtained from OPT in Claim 3, LR the set of edges selected by the local ratio approximation algorithm when applied to the U-Steiner instance, $k(LR)$ the number of connected components of LR which intersect A, and $m(LR)$ the number of moves of the solution obtained in Claim 3 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:

$$
\begin{aligned}
m(LR) &\leq |LR| + k(LR) - c \quad \text{from Claim 3} \\
&= \delta(LR) + k(LR) - c \\
&= \alpha\delta_1(LR) + \delta_2(LR) + k(LR) - c \\
&\leq \alpha(2\delta_1(OP) - k(LR)) + \delta_2(LR) + k(LR) - c \quad \text{by Claim 4} \\
&\leq 2\alpha\delta_1(OP) - k(LR) + \delta_2(LR) + k(LR) - c \quad \text{since } \alpha \geq 1 \\
&\leq 2\alpha\delta_1(OP) + 2\delta_2(OP) - c \quad \text{by Equation 1} \\
&\leq 2\delta(OP) - c \\
&= 2|OP| - c \\
&\leq 2(m(OPT) + c) - c \quad \text{from Claim 3} \\
&= 2m(OPT) + c \\
&\leq 3m(OPT),
\end{aligned}
$$

since every solution must move the chips of C. This finishes the proof of Theorem 3. □

Remark. ADRIAN SAYS: To be inserted: on the graph version, when moves are along edges: can be solved in polyn. time by matching; some details to follow. ⟵

3 Chips in grids

In this Section we analyze the reconfiguration problem with labeled, respectively unlabeled chips, in an infinite grid. However a finite section of the grid clearly suffices for this purpose.

3.1 Hardness Results

Theorem 4 *The unlabeled version* 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 asks 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 [10] and thus we can assume all the coordinates are given in unary. Consider an instance of R-STEINER with one of the points at the origin and all the other $n-1$ points in the first quadrant. The instance of U-GRID-RP is illustrated in Fig. 3. It has $n-1$ chips located on the x-axis at coordinates $0, -1, -2, \ldots, -n+1$, and $n-1$ targets located at the the points in the first quadrant. Consider a sufficiently large axis-aligned rectangle enclosing the chips and targets with obstacles placed at each other integer point (the boundary of the rectangle at distance $\Delta_x + \Delta_y$ from any chip or target, where Δ_x, Δ_y are the dimensions of the smallest axis-aligned rectangle enclosing the chips and targets). 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 (proof omitted).

Claim 5 *There is a rectilinear Steiner tree of length at most q if and only if reconfiguration can be done using at most q moves.*

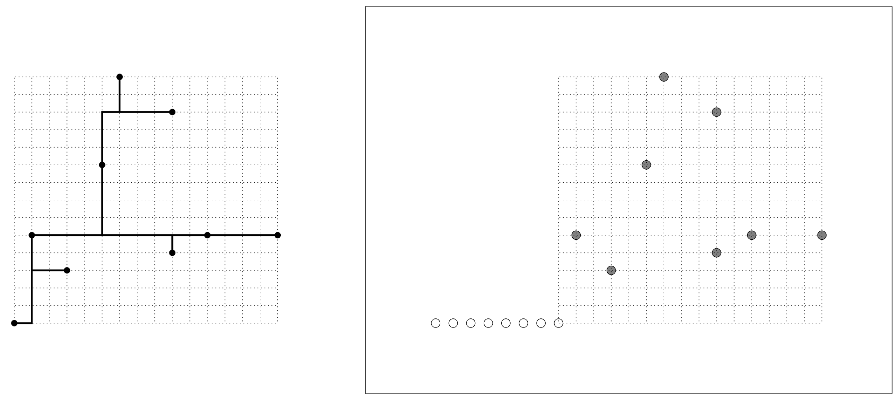

Figure 3: Unlabeled chips in the grid obtained from a Rectilinear Steiner Tree instance with $n = 9$ (sketch). Obstacles are placed at each other integer point in the axis-aligned bounding rectangle.

□

Remark. Conform with the results of [3], there exists a polynomial-time approximation scheme for Rectilinear Steiner Tree, and we don't expect an inapproximability result for U-GRID-RP to hold.

Theorem 5 *The labeled version* L-GRID-RP *is NP-complete.*

Proof. The argument, similar to that in the proof of Theorem 4, 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. 4. It has n labeled chips $s_1, \ldots, s_n$ located at the

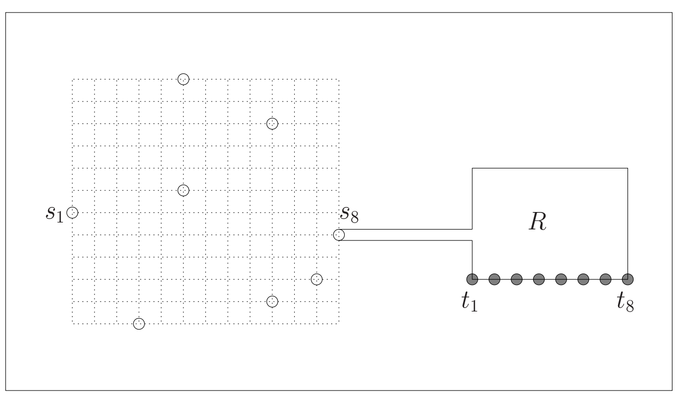

Figure 4: Labeled chips in the grid obtained from a Rectilinear Steiner Tree instance with $n = 8$ (sketch). Obstacles are placed at each other integer point in the axis-aligned bounding rectangle. The corridor and most of the target area R (except its bottom row) are free.

points in P (s_i at p_i). Consider a sufficiently large axis-aligned rectangle enclosing the chips and filled with obstacles placed at each other integer point, except the following: if 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 chips; both the corridor and all rows of R except its bottom row are free. The bottom row contains all n targets $t_1, \ldots, t_n$; 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 (proof omitted).

Claim 6 *There is a rectilinear Steiner tree of length at most q if and only if reconfiguration can be done using at most $2q + 2$ moves.*

□

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

Theorem 6 *Given a pair of start and target configurations S and T, each with n labeled chips, one can move chips from S to T using at most $7n/4$ moves. On the other hand, $3n/2$ moves are sometimes necessary (n even).*

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 $S \to T$. 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 partioned 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 odd (resp. even) rows; we can assume without loss of generality that $|D| \le |E|$, thus $|D| \le n/2$. Let A be the set of elements of E whose target lie in E, and let B (resp. C) be the 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 $c \le b$.

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 remaining among the elements of A (not from the loops).

4. Fill 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.

6. Fill 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 [6, 7] for related aspects of disk reconfiguration problems).

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

References

[1] M. Abellanas, F. Hurtado, A. G. Olaverri, D. Rappaport, and J. Tejel, Moving coins. Short version in *Abstracts of Japan Conference on Discrete and Computational Geometry*, 2004. Full version submitted to *Computational Geometry: Theory and Applications*.

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

[3] S. Arora, Nearly linear time approximation schemes for Euclidean TSP and other geometric problems, *J. of the ACM*, **45**(5):1–30, 1998.

[4] 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.

[5] R. Bar-Yehuda, One for the Price of Two: A Unified Approach for Approximating Covering Problems, *Algorithmica*, **27** (2000), 131–144.

[6] S. Bereg, A. Dumitrescu, and J. Pach, Sliding disks in the plane, *Japan Conference on Discrete and Computational Geometry 2004* (J. Akiyama, M. Kano and X. Tan, editors). To appear in LNCS Proceedings.

[7] S. Bereg and A. Dumitrescu, The lifting model for reconfiguration, *Proceedings of the 21st Annual Symposium on Computational Geometry*, (SOCG '05), Pisa, Italy, June 2005, 55–62.

[8] A. Dumitrescu and J. Pach, Pushing squares around, *Graphs and Combinatorics*, accepted. A preliminary version in *Proceedings of the 20-th Annual Symposium on Computational Geometry*, (SOCG '04), NY, June 2004, 116–123.

[9] A. Dumitrescu, I. Suzuki and M. Yamashita, Motion planning for metamorphic systems: feasibility, decidability and distributed reconfiguration, *IEEE Transactions on Robotics and Automation*, **20(3)**, 2004, 409–418.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

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