Editors' Suggestion

Enhanced atom-by-atom assembly of arbitrary tweezer arrays

Kai-Niklas Schymik, Vincent Lienhard, Daniel Barredo, Pascal Scholl, Hannah Williams, Antoine Browaeys, and Thierry Lahaye
Université Paris-Saclay, Institut d'Optique Graduate School, CNRS, Laboratoire Charles Fabry, 91127 Palaiseau, France

(Received 13 October 2020; accepted 20 November 2020; published 10 December 2020)

We report on improvements extending the capabilities of the atom-by-atom assembler described by Barredo *et al.* [Science **354**, 1021 (2016)] that we use to create fully-loaded target arrays of more than 100 single atoms in optical tweezers, starting from randomly loaded, half-filled initial arrays. We describe four variants of the sorting algorithm that decrease the number of moves needed for assembly and enable the assembly of arbitrary, nonregular target arrays. We demonstrate experimentally the performance of this enhanced assembler for a variety of target arrays.

DOI: 10.1103/PhysRevA.102.063107

I. INTRODUCTION

Over the past few years, arrays of single laser-cooled atoms trapped in optical tweezers have become a prominent platform for quantum science, in particular for quantum simulation [1]. They allow single-atom imaging and manipulation, fast repetition rates, and high tunability of the geometry of the arrays. When combined with excitation to Rydberg states, these systems naturally implement quantum spin models, with either Ising [2–6] or XY [7] interactions. They can also be used to realize quantum gates with fidelities approaching those of the best quantum computing platforms [8–11].

A crucial ingredient of the atom array platform is the atom-by-atom assembly of fully loaded arrays, starting from the partially loaded arrays (with a typical filling fraction of 50%–60%) obtained when loading optical tweezers with single atoms [12]. This technique, first demonstrated in [13–15], can follow different approaches. A fast and effective approach for realizing one-dimensional chains uses an acousto-optic deflector (AOD) driven with multiple radio-frequency tones to generate all the traps [14]; after loading, empty traps are then switched off and the remaining ones are brought to their target position, thus achieving a fully loaded chain in a single step. However, directly extending this approach to more than one dimension is challenging [16]. A different approach consists in using a spatial light modulator (SLM) to generate arbitrary patterns of traps in one, two, or three dimensions, load them with atoms, and then dynamically change the SLM pattern to rearrange the atoms in space [17]. However, SLMs are slow, making the rearrangement time prohibitive, which limits this approach to small atom numbers. Another approach is using a static trap array and combining it with a moving tweezer [13,18].

Our experiment [13] follows this strategy and uses an SLM that produces a user-defined fixed pattern of optical tweezers which includes the final (target) array, combined with a moving tweezer. This extra microtrap, controlled by a two-dimensional (2D) AOD, is used to move the atoms one by one to reach a fully loaded target array. The heuristic shortest-moves-first algorithm used in [13] to find the set of needed moves is versatile, as any target array included in an initial regular array can be assembled. It works well up to a few tens of atoms, but it has some limitations. First, the algorithm was written for regular arrays, such as square and triangular lattices. On completely arbitrary arrays, lattice edges along which atoms can be moved are not naturally given, and using straight paths between source and target traps would lead to unwanted losses, as another target trap already containing an atom may be in the way. Another limitation is that the number of moves needed for ordering is not optimal and minimizing this number becomes more crucial when the number N of assembled atoms increases beyond a few tens.

Here we describe four improved algorithms that can be used without any change in the hardware; the choice of the most efficient approach depends on the characteristics of the target array. We first recall in Sec. II the problem we need to solve and review our previous approach and its shortcomings (Sec. III). We then discuss in Sec. IV a compression algorithm which is well adapted for compact arrays (here, by compact we mean that no trap other than target ones lies within the target array). The number of moves is then at most N, which significantly reduces the assembly time. We show in Sec. V that a similar scaling can be obtained for all arrays (compact or not) by using algorithms based on a linear sum assignment problem solver. In Sec. VI we extend these algorithms to the case of fully arbitrary two-dimensional patterns (i.e., they are not embedded in a regular Bravais lattice). Finally, in Sec. VII we experimentally implement these approaches in a variety of arrays.

II. STATEMENT OF THE PROBLEM

Our goal is to obtain a fully loaded array of N traps, whose positions are given by the user (this defines the *target* array, denoted by green circles in this paper). To do so, we start from a larger array, with $\sim 2N$ traps, containing the target array and extra, *reservoir* traps (these will be denoted by red circles). The entire array is loaded in a stochastic way with an $\sim 50\%$ filling fraction at each realization of the experiment.

Therefore, we have, with high probability, at least N atoms in the full array. Using a moving optical tweezer, we then transport the atoms one by one, from an initial trap to one of the target traps, until the target array is fully filled.

To maximize the success probability of the assembly process, we need to minimize the total assembly time. One reason for that arises from the vacuum-limited lifetime of a trapped atom, which, in our experiments, is $\tau_{\rm vac} \sim 20$ s. This means that for an array with N atoms, the lifetime of the configuration is $\tau_{\rm vac}/N$. It is thus important, when N increases, to minimize the total assembly time to reduce atom losses during rearrangement. As atoms are moved between traps at a constant velocity (typically $\sim$100 nm/μs, meaning we need $\sim$50 μs to move over a typical nearest-neighbor distance of 5 μm) and as it requires a comparatively longer time (600 μs) to capture or release an atom [13], minimizing the arrangement time mainly amounts to minimizing the number of moves and, but to a lesser extent, the total travel distance (defined as the sum of the lengths of the successive straight paths over which an atom is moved). A second reason for minimizing the number of moves is that each transfer from a source trap to a target trap has a finite success probability p (typically around $p \sim 0.98$–0.99 in our experiments), partly due to the already mentioned vacuum-limited losses, but also due to, e.g., inaccuracy in the positioning of the moving tweezers or residual heating. Beyond the number of moves and the total travel distance, the time it takes for the algorithm to compute the moves at each repetition of the experiment contributes to the total assembly time.

In [13] we distinguished two types of moves for reordering. The first approach (which we called type 1) corresponds to the situation where the atom can be moved in between adjacent rows of traps. Then, as on average $N/2$ atoms are out of place initially, the mean number of needed moves is $N_{\rm mv} = N/2$ and we have to solve a linear sum assignment problem [19]. Using the Hungarian algorithm (as in [20]) then minimizes the assembly time. However, type-1 moves require a large distance (at least 5 μm) between any two traps to avoid atom loss due to disturbances of the trap potential. In practice, many experimental reasons (the finite field of view of the lenses used to focus the tweezers, the need to have large interaction strength between Rydberg atoms, and to have uniform Rydberg excitation lasers over the array) call for having smaller distances in the arrays. Furthermore, as we will see in Sec. VI, type-1 moves are not well suited for the assembly of truly arbitrary geometries. For these reasons, we here focus on solving our problem using just type-2 moves, where an atom can only be moved along a straight path between adjacent traps.

In the case of type-2 moves, assigning any source trap to any target trap is not possible, since other traps might be in the way. While an atom can be moved over an empty trap as the moving tweezer is $\sim$10 times as deep as the stationary traps, having filled traps on the path would lead to collisions and atom loss. Finding the optimal set of moves is thus nontrivial since it requires finding a collision-free assignment with a well-defined ordering of the moves. In computer science, this problem is known as the pebble motion on a graph (in a variant with unlabeled pebbles) and is intractable for large N [21], even more so in practice as we need to solve it in a time short compared to the configuration lifetime. Therefore, we opt for heuristic algorithms, provided they give a solution not too far from the optimum and run in a few tens of milliseconds at most for up to a few hundred atoms. In the next section, we will see that the algorithm used in [13] actually meets these criteria only when the target array is not too compact and when N is not too large.

III. OUR PREVIOUS ASSEMBLER: PRINCIPLE OF OPERATION AND LIMITATIONS

The atom-by-atom assembler described in [13,22] allowed us to create user-defined arrays in one, two, and three dimensions at unit filling. Nonperiodic structures, or complex lattices such as ladder, honeycomb, kagome, or pyrochlore geometries could also be obtained by selecting a subset of target traps on an underlying Bravais lattice.

We chose a heuristic approach to the problem that had the advantage of requiring a short computation time, scaling as $O(N^2)$, albeit at the expense of not guaranteeing the optimal assignment. This greedy algorithm, which we will call the shortest-moves-first algorithm, works as follows. We first compute a matrix of distances $D = d_{ij}$ between each out-of-place atom s_i and each (empty) target t_j trap. Then we order the entries of this matrix by increasing length and select the first $N/2$ elements with the condition that only one element per row or column is chosen (i.e., that each atom or target trap is only assigned once).

This initial matching is not collision-free, as already filled traps may lie in between a matched reservoir atom and an empty target trap. Therefore, in a second step, we postprocess this assignment by applying a rule that splits each move $S \rightarrow T$ from a source atom S to a target trap T in two moves $O \rightarrow T$ and $S \rightarrow O$ for each obstacle atom O that is found in the path. Note that this process leaves the travel distance unchanged but increases the number of moves, therefore increasing the total assembly time.

Figure 1 shows the number of moves $N_{\rm mv}$ returned by the above algorithm to assemble a target array of N atoms embedded in a square array, for three different geometries: (i) a staggered pattern, (ii) a random pattern, and (iii) a compact square in the center. The number of moves is averaged over 1000 realizations of the initial random loading. We observe that $N_{\rm mv}$ is only slightly above $N/2$ for the cases (i) and (ii) where the reservoir and target traps are strongly mixed. However, in the case (iii) of compact geometries, where all the reservoir atoms lie outside the target array, we observe that this procedure scales as $N_{\rm mv} \propto N^{\alpha}$, with $\alpha \simeq 1.4$ (red dashed line), making it unsuitable for large arrays.

The reason for this is illustrated in Fig. 2, which shows a few snapshots of the reordering process. The shortest moves are those connecting out-of-place atoms with target traps on the border of the array; therefore, the algorithm starts by filling the outermost shell. Once this is done, it is no longer possible to fill the (empty) inner traps without performing extra operations to displace the atoms that lie in the way, giving rise to many extra moves to fill the inner part of the target array. For the initial configuration in Fig. 2(b), the 14×14 target array is assembled in 444 moves. As picking up and releasing an atom takes extra time, this behavior leads to prohibitive rearrangement times, even if the distance traveled is close to optimal.

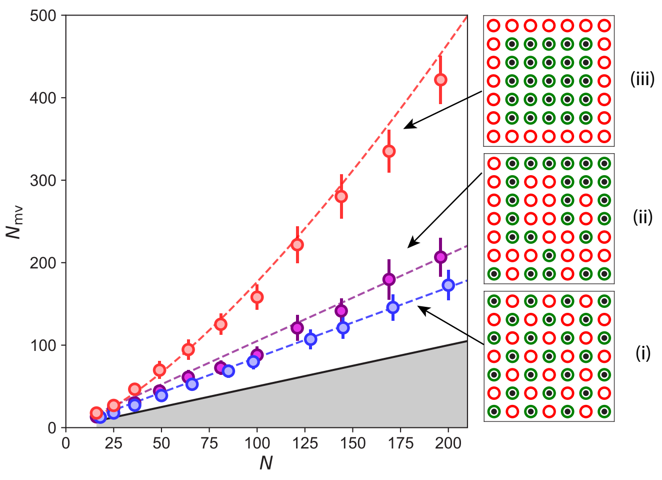

FIG. 1. Scaling of the number of moves for different geometries, with the shortest-moves-first approach. The plot shows the number N_{mv} of moves (averaged over 1000 realizations of the random loading; the error bars indicate the standard deviation of the distribution of N_{mv}) as a function of the size N of the target array. For staggered configurations (blue), where a target trap and a reservoir trap alternate, the overhead as compared to the lower bound $N/2$ (indicated by the solid black line above the gray-shaded area) is small. For a random subset of target traps in a square array (purple), the number of postprocessing moves due to obstacles is already bigger, but the scaling is still linear with N. A drastic change appears in the case of compact geometries (red), where the target array is surrounded by reservoir atoms. Here the number of moves does not increase linearly with N, but rather as $N^{1.4}$ (dashed line) and many postprocess moves are needed. This means that the current algorithm is unsuited for large compact geometries.

This behavior is problematic, as many arrays of interest for quantum simulation are compact. Therefore, it is crucial to find an assignment between the reservoir and target traps which really minimizes the number of moves. For assembling compact arrays, a much better approach, where the maximum number of moves is at most N, is the compression algorithm that we now describe.

IV. IMPROVED ASSEMBLY OF COMPACT ARRAYS BY THE COMPRESSION ALGORITHM

From the above considerations it is clear that we need to prevent the formation of the outer shell during the assembling process. A simple way to do this and have a collision-free assignment without any postprocessing is to fill the target traps in a determined order. We first fill the central traps and progressively, one layer after the other, we fill the compact structure until we reach its border. To fill the traps, we choose the closest atoms lying outside the already assembled bulk. An asset of this compression approach is that we can calculate once, independently of the initial loading, a lookup table. The table stores which source traps can be used to fill a given target trap. In combination with the predetermined order in which the target traps are filled, the lookup table reduces the calculation time of a particular instance. We observe that it scales roughly as $N^{1.2}$ with the number of target traps and amounts, in our implementation, to about 7 ms for $N = 100$ on a regular desktop computer with 16 GB of RAM.

Figure 3(a) illustrates how the algorithm works on a small square array. The target array is first assembled from the bottom left corner, then the diagonal, and finally the top right corner. Using this algorithm, atoms which initially occupy target traps can be displaced, which means additional moves with respect to an optimal solution. However, as we always use the atoms closest to the border of the compact structure to assemble it, the path is always obstacle-free and therefore we do not need any postprocessing. Consequently, each atom is moved *at most once* during the assembling process, which sets the upper bound $N_{\text{mv}} \leqslant N$ and ensures on average a smaller number of moves using the compression algorithm with respect to the shortest-moves-first algorithm of the preceding section. As N_{mv} cannot be lower than $N/2$ on average, our solution, while not optimal for many initial loading instances, is generally close to optimal. Figure 3(b) shows how this

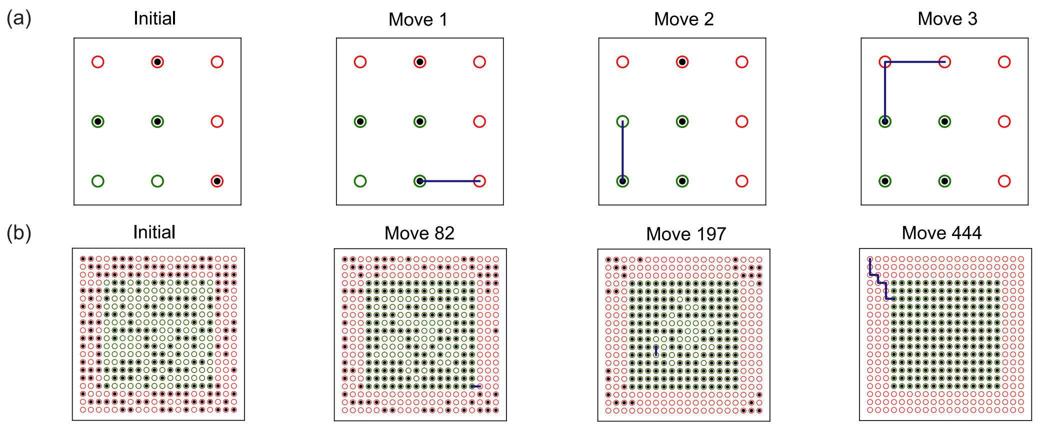

FIG. 2. Assembling of a compact array using the shortest-moves-first algorithm. (a) Microscopic view. The first set of moves (blue lines) connects out-of-place atoms with target traps on the outer shell of the structure (e.g., move 1). Once the border is populated, it is no longer possible to fill the inner traps without performing extra moves (move 2). (b) The macroscopic behavior on a 14×14 array reveals that the algorithm starts by filling the border of the target array (green circles) with atoms from reservoir traps (red circles), while inner traps are still empty (e.g., move 82), leading to a large overhead in the number of moves for successful assembling.

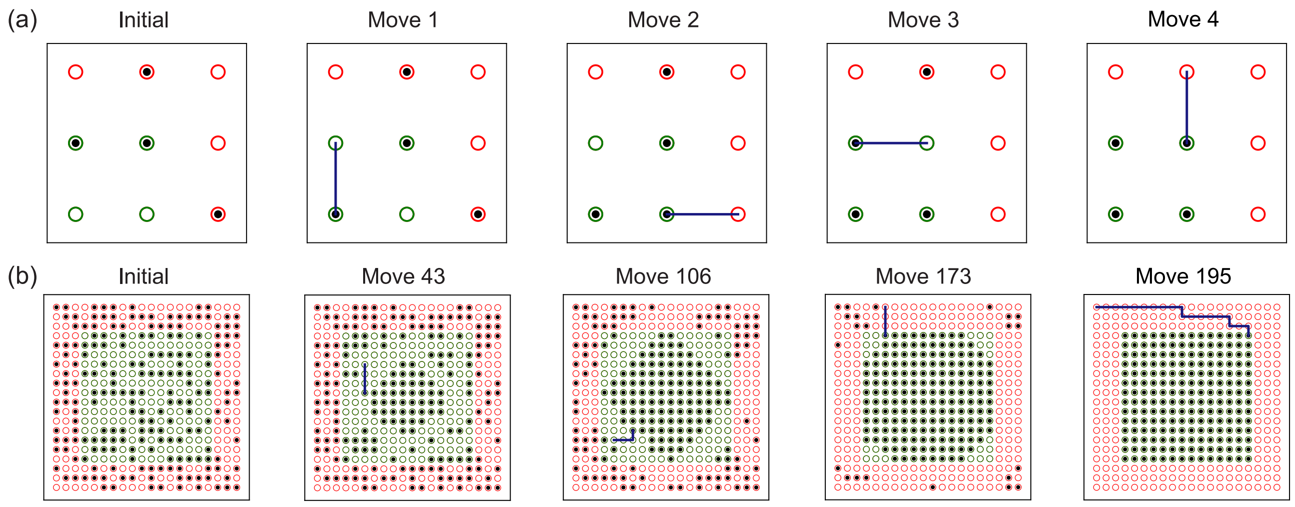

FIG. 3. Compression algorithm. (a) Illustration of the compression procedure for a 2×2 target array, requiring four moves. (b) A few assembling steps using the compression algorithm to assemble a 14×14 target array in only 195 moves, to be contrasted with the 444 moves needed previously.

compression algorithm outperforms the shortest-moves-first one. The 196 target atoms are assembled in 195 moves, whereas the same initial configuration required 444 moves to be sorted with our previous approach.

As can be seen in Fig. 4(a), not only is the average number of moves smaller than before, but the distribution of $N_{\rm mv}$, calculated for 1000 random initial loading instances of the array, is also strongly sub-Poissonian, as well as asymmetric, with a sharp cutoff at N. This is an appealing feature, as it indicates that the success probability of the assembly process should be more consistent from one shot to another, as compared to the previous approach. Figure 4(b) shows the linear scaling of $N_{\rm mv}$ with N.

This technique can be naturally extended to the case of compact structures in other lattices (e.g., triangular) and also to arbitrary geometries, as we will see in Sec. VI.

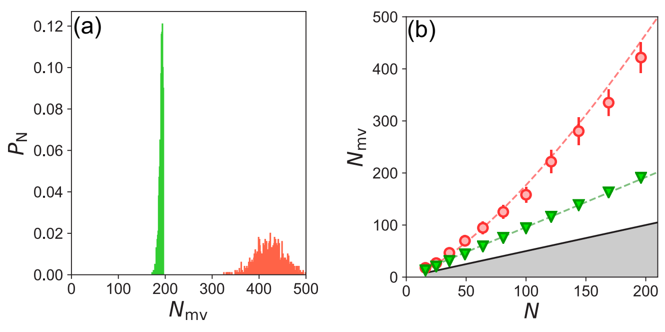

FIG. 4. Compression vs shortest-moves-first algorithms. (a) Histogram of the number of moves needed to fill a 14×14 square array for 1000 initial random loading instances. The compression algorithm (green) has a narrow distribution which is bounded by N. The shortest-moves-first algorithm (red) has a broad distribution and requires on average many more moves since the initial assignment is not collision-free. (b) Comparison of the scaling of $N_{\rm mv}$ as a function of N between the two algorithms. The compression algorithm gives a number of moves linear in N. Error bars are the standard deviation of the distribution.

V. USING A LINEAR SUM ASSIGNMENT PROBLEM SOLVER

In view of minimizing the number of moves, it is interesting to revisit the approach of the problem as a linear sum assignment problem (LSAP), which was mentioned above for the case of type-1 moves. However, for the type-2 moves we are interested in here, a direct application of the LSAP matching with the travel distance ℓ as a cost function does not yield a collision-free assignment and requires postprocessing, which in general increases the number of moves. We describe in this section two different algorithms that first use a LSAP solver and then reprocess the moves, which leads to a low number of moves. The LSAP solver we use in practice is a modified Jonker-Volgenant algorithm with no initialization [23], which is implemented in the `scipy.optimize` PYTHON package [24].

The first algorithm (LSAP1) uses the total travel distance $\sum_{\text{moves } i} \ell_i$ as the cost function, while the second one (LSAP2) uses a modified metric $\sum_{\text{moves } i} \ell_i^2$, which favors shorter paths [Fig. 5(a)]. In both cases, the set of returned moves is postprocessed to eliminate collisions and reduce the number of moves.

A. LSAP1: Standard metric and merging

Our first approach, described using a simple example in Fig. 5(b), starts with the LSAP algorithm using the travel distance between the source and target traps as a cost function. We first sort the returned moves from shortest to longest. Since the found assignment leads to collisions, we then postprocess the set of moves by splitting the paths with obstacles into two or more moves, just as in the shortest-moves-first approach. However, in a second iteration, we merge again some moves in which an atom is picked up twice, thereby reducing the number of moves considerably, checking at each step that we do not reintroduce any collision in doing so. Note that this second merging iteration can in principle be applied to any algorithm, but yields the smallest number of moves when

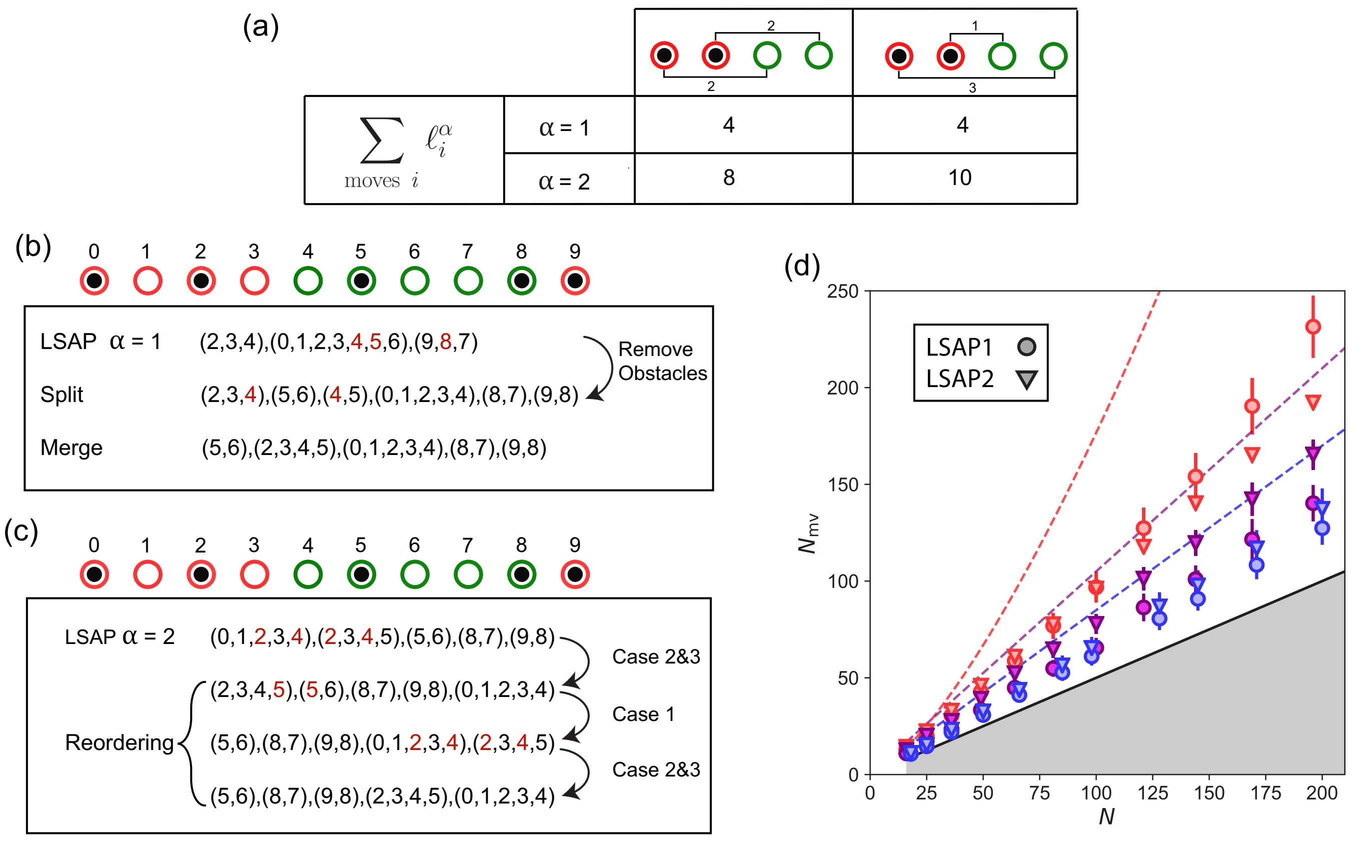

FIG. 5. Modified LSAP algorithms. (a) Using a cost function with $\alpha = 2$ (see the text) in a LSAP solver favors short moves. (b) The algorithm LSAP1 first uses a LSAP solver with $\alpha = 1$, which returns a list of moves [here (2,3,4) means that the atom initially in trap 2 is moved, via trap 3, to trap 4]. Some moves lead to collisions (denoted in red) and thus the set of moves is postprocessed as in the shortest-moves-first algorithms, by splitting the problematic moves into two or more stages. However, in a second step, two moves that share the same trap as final and initial positions (denoted in red) can be merged together, reducing the total number of moves. (c) The algorithm LSAP2 uses a modified cost function with $\alpha = 2$, which returns a set of short moves; to avoid collisions, the moves are then reordered by applying successively three rules (see the text) until the rearrangement can be performed without collisions. Numbers in red highlight the breaking of a rule. (d) Number N_{mv} of needed moves as a function of N to assemble a staggered target array (blue), a random target array (purple), or a compact target array (red), for the LSAP1 and LSAP2 algorithms. The dashed lines reproduce the fits of Fig. 1 for comparison.

starting from the LSAP matching. The computation time for this approach is on average 4 ms for 100 target traps in a staggered geometry and roughly scales as N^2.[1],[2]

Figure 5(d) shows the number of moves N_{mv} as a function of N for LSAP 1 (disks). The performance is very satisfactory for staggered or random target arrays, as the number of moves is only 20–30 % higher than the absolute lower bound $N/2$. For compact arrays, the number of needed moves is slightly larger than N, making this approach less efficient than the compression algorithm described in Sec. IV.

[1]In the worst case, the Hungarian matching algorithm is known to scale as N^3; however, we observe empirically that for the current problem and for the values of N up to a few hundreds considered here, the average runtime of our LSAP and reordering algorithm scales roughly as N^2.

[2]To reduce the computation time during the experiment, we precalculate a lookup table with the shortest paths and path lengths between all trap pairs. During each assembly cycle, the cost matrix for the LSAP algorithm is found as a submatrix of the lookup table.

B. LSAP2: Modified metric and reordering

Long moves lead to many collisions; therefore, it is beneficial to avoid them. In our second approach we achieve this by using a modified cost function $\sum_{\text{paths } i} \ell_i^2$. A similar idea was introduced in [20], but here the moves are sequential and we thus need to find the right ordering in which the moves have to be performed to avoid collisions.

To do so, we apply the following rules. We examine each move in the list and if the target trap of the move is occupied (case 1), if another trap along the path of the move is filled (case 2), or if the target trap is in the path of another move following in the list (case 3), we postpone this move and put it at the end of the list of moves. We find empirically that this procedure always produces a collision-free set of moves. This approach is illustrated in Fig. 5(c). The whole algorithm (LSAP and reordering) has an average computation time of 4 ms for $N = 100$ target traps in a compact geometry and scales roughly as N^2.

Whatever the target array, the maximum number of moves is bounded by N, the size of the cost matrix. As can be seen in Fig. 5(d) (triangles), the number of moves returned by

LSAP2 is slightly larger than LSAP1 for sparse arrays, but is smaller for compact arrays, where it gives essentially the same performance as the compression algorithm. The latter, however, has the advantage of a shorter calculation time for $N > N_c$, with a critical atom number $N_c \sim 300$ in our current implementation.

VI. ARRAYS WITH COMPLETELY ARBITRARY GEOMETRY

Condensed-matter models are often studied on specific crystalline arrangements which are described by a Bravais lattice, e.g., a square or a triangular lattice. Our previous assembler was therefore based on such an underlying lattice, which simplifies the problem in two ways. First, this naturally defines the paths along which the moving tweezer can travel and, because these lattice edges are separated by a constant spacing, it automatically ensures that a minimal distance between atoms in traps and the moving tweezer is always kept during the rearrangement. Second, it simplifies the distance calculation between two traps by defining the metric in terms of lattice coordinates (Manhattan distance).

Not all physical structures of interest for quantum simulation, however, can be described by a Bravais lattice. Examples of such nonperiodic features include crystals with defects (interstitial defects, vacancies, dislocations, and grain boundaries), quasicrystals, disordered arrays for Anderson or many-body localization studies, and even totally arbitrary structures in the context of combinatorial optimization problems such as finding the maximum independent set of a graph [25,26]. To examine such systems, we developed a variant of our algorithms, which is not based on an underlying lattice and therefore allows us to assemble truly arbitrary structures.

The starting point for our algorithm is the set of N target traps, whose positions are provided by the user. Because of the stochastic loading, we have to place N additional reservoir traps close to the arbitrary N-atom target configuration. This reservoir generation works as follows [Fig. 6(a)]. Whenever possible, to reduce the number of moves, a reservoir trap should be placed in immediate proximity to each target trap. To do so, we compute the Voronoi diagram [27] of the set of target traps (i.e., divide the plane in N regions, one around each target trap T, such that all points of this region are closer to T than to any other trap). We then add in each Voronoi cell a single reservoir trap, provided it can be placed at a distance larger than a "safety" distance d_m (typically $\sim$4 μm) from all other traps. If successful, this procedure ensures that for each target trap there is a single reservoir close to it [Fig. 6(b)]. If, however, the density of the target traps is already comparable to $1/d_m^2$, then we cannot add enough reservoir traps in this way and so we place extra traps at the periphery of the pattern in a compact triangular array [Fig. 6(c)].

The next step is to find paths along which an atom can travel to an empty target trap. Contrary to the case of Bravais lattices, no obvious edges are *a priori* connecting the traps along which the moves can be performed. Direct straight-line paths from the reservoir to the target trap are also not possible, since there can be other traps in the way, leading to collisions and atom losses. We thus define the set of allowed paths by using a Delaunay triangulation [27] of the full set of traps

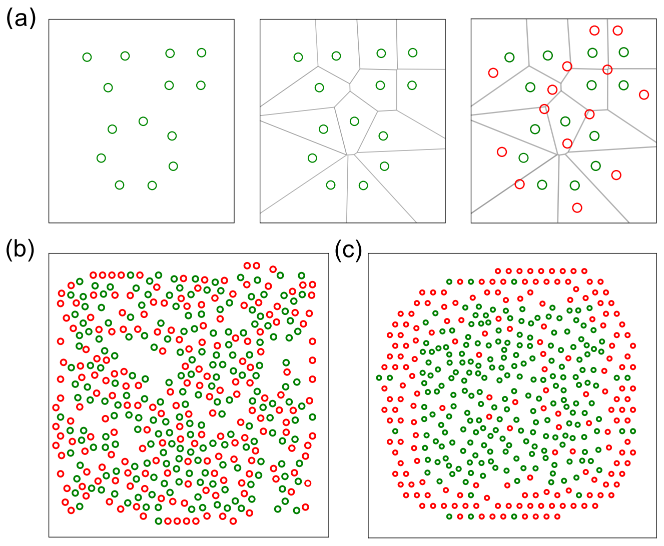

FIG. 6. Generating the reservoir arrays for arbitrary target arrays. (a) Starting from the user-defined target array (left), we compute its Voronoi diagram (middle) and in each cell we add a reservoir trap, shown in red, if there is enough room (right); otherwise we add it at the periphery (see the text for details). Also shown are examples of generated reservoirs for an $N = 200$ target array, (b) without and (c) with the need to add reservoirs at the periphery.

(target and reservoir) as shown in Fig. 7. In practice, we implemented the triangulation in PYTHON 3.0 with the SCIPY library [24]. To enforce the above-mentioned constraint of a minimal passing distance, we postremove edges that do not meet this requirement (see dashed lines in Fig. 7). We emphasize that the generation of the reservoir traps and of the allowed edges is done just once for any given target array, and not at each repetition of the experiment, which considerably relaxes the constraints on the speed of this algorithm. In practice, arrays with hundreds of target traps can be processed in a few seconds.

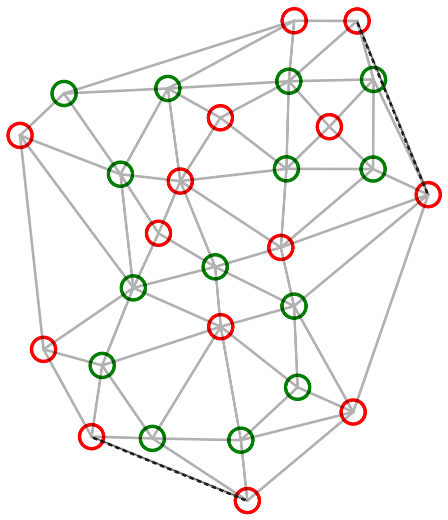
FIG. 7. Generating the allowed paths between traps. We first perform the Delaunay triangulation of the atom array. In a second step, we remove edges which do not fulfill a minimal passing-distance requirement (dotted line).

This triangulation then allows us to naturally describe the whole structure in terms of graph language, connecting the nodes (trap positions) by edges along which the atoms are allowed to move. In this way, we eliminate the necessity to describe the problem with an underlying Bravais lattice. Furthermore, it allows the implementation of efficient shortest-path graph algorithms (e.g., the Dijkstra algorithm [19]) to find the shortest path between a matched initial and target trap, following the allowed edges of the graph. For the generation of the graphs and graph algorithms the NETWORKX library [28] is used. With these modifications, it is now possible to extend the algorithms discussed above to arbitrary patterns. The scaling and performance of the algorithms (in terms of computation time and the number of moves) are essentially unchanged as compared to the case of regular lattices.

VII. EXPERIMENTAL DEMONSTRATION

The experimental setup has been described in [13]. Using an SLM (Hammamatsu X10468-02), a fixed pattern of optical dipole traps at 850 nm is generated in the focal plane of a high-numerical-aperture (equal to 0.5) aspheric lens. With an available laser power of $\sim$1 W, we can generate up to 200 traps with a $1/e^2$ radius of $\sim$1 μm and a typical trap depth of $\sim$1 mK, resulting in a radial (longitudinal) trapping frequency around 100 kHz (20 kHz). Initially, the traps are stochastically loaded with single atoms at a temperature of $\sim$10 μK from a magneto-optical trap of ^{87}Rb atoms; the typical loading time is $\sim$150 ms. An initial fluorescence image (20 ms) determines the initial occupancy of the traps, which is 50–60% on average.

To assemble a target array, we use a single 850-nm dipole trap with a $1/e^2$ radius of $\sim$1.3 μm, steered by a 2D AOD, which can pick up an atom from a static trap by ramping up its depth to $\sim$10 mK and subsequently moving and then releasing the atom at the position of an empty static trap. After the assembly, a fluorescence image with an exposure time of 20 ms determines the occupancy of the target array, before we perform an actual experiment, e.g., quantum simulation of a spin model, by exciting the atoms to Rydberg levels [1]. This technique allows us to perform experiments with a typical repetition rate of $\sim$3 Hz.

Once the trap array has been generated, we equalize the trap intensities using the fluorescence signal of the loaded traps.[3] Then the choice of the optimal algorithm to be used for assembly, among the three described above, is made according to the characteristics of the target array to assemble, as described in Fig. 8.

Finally, in order to further improve the success probability of assembling a defect-free array, we apply multiple

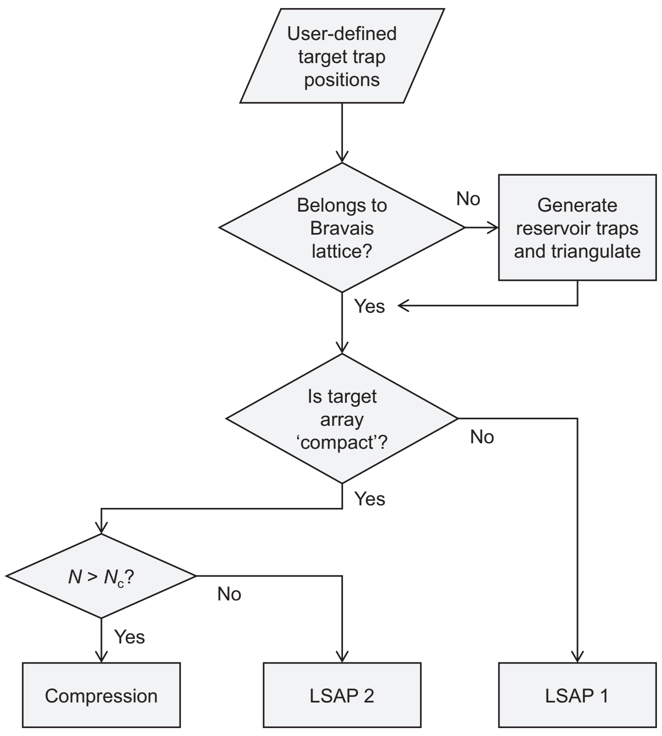
User-defined target trap positions
Belongs to Bravais lattice?
No
Generate reservoir traps and triangulate
Yes
Is target array 'compact'?
No
Yes
$N > N_c$?
No
Yes
Compression
LSAP 2
LSAP 1

FIG. 8. Algorithm choice flowchart. The best-suited algorithm to be used depends on the characteristics of the target array. The critical atom number N_c is defined at the end of Sec. V.

rearrangement cycles (similar to [14,18]). At the end of the first rearrangement process, we keep the excess atoms and determine the defects with a fluorescence image. We then fill these defects [Fig. 9(a)]. This process can be repeated until a defect-free array is obtained and excess atoms are removed. However, since this procedure requires more than N initial

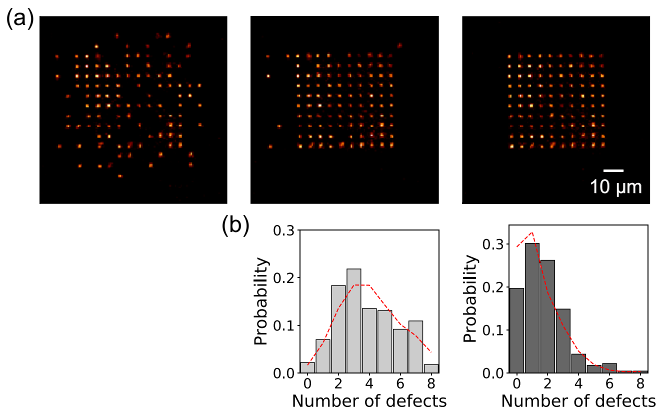

FIG. 9. Multiple rearrangement cycles. The probability to assemble a defect-free array can be increased by starting with more than N atoms and repeating the rearrangement cycle more than once. (a) On the shown 10×10 compact target square array, we can increase the probability to create a defect-free array by a factor 10 (from 2% to 20%), when starting with 225 atoms and performing a second cycle. (b) A Monte Carlo simulation (red) of the first cycle and second cycle, including the measured efficiencies of performing the moves and vacuum lifetime, reproduces the experimental distribution of defects reasonably well.

[3]It is of importance that all microtraps have a good optical quality and in particular the same depth such that (i) single-atom loading does indeed occur with a probability of $\sim$1/2 and (ii) the fluorescence signal from any given trap allows for efficient identification of the presence of a single atom. We now equalize the trap depths by a direct optimization of the fluorescence time trace of each single trap, altering the trap intensity until we fulfill criteria (i) and (ii). A detailed description of this procedure is left for future work.

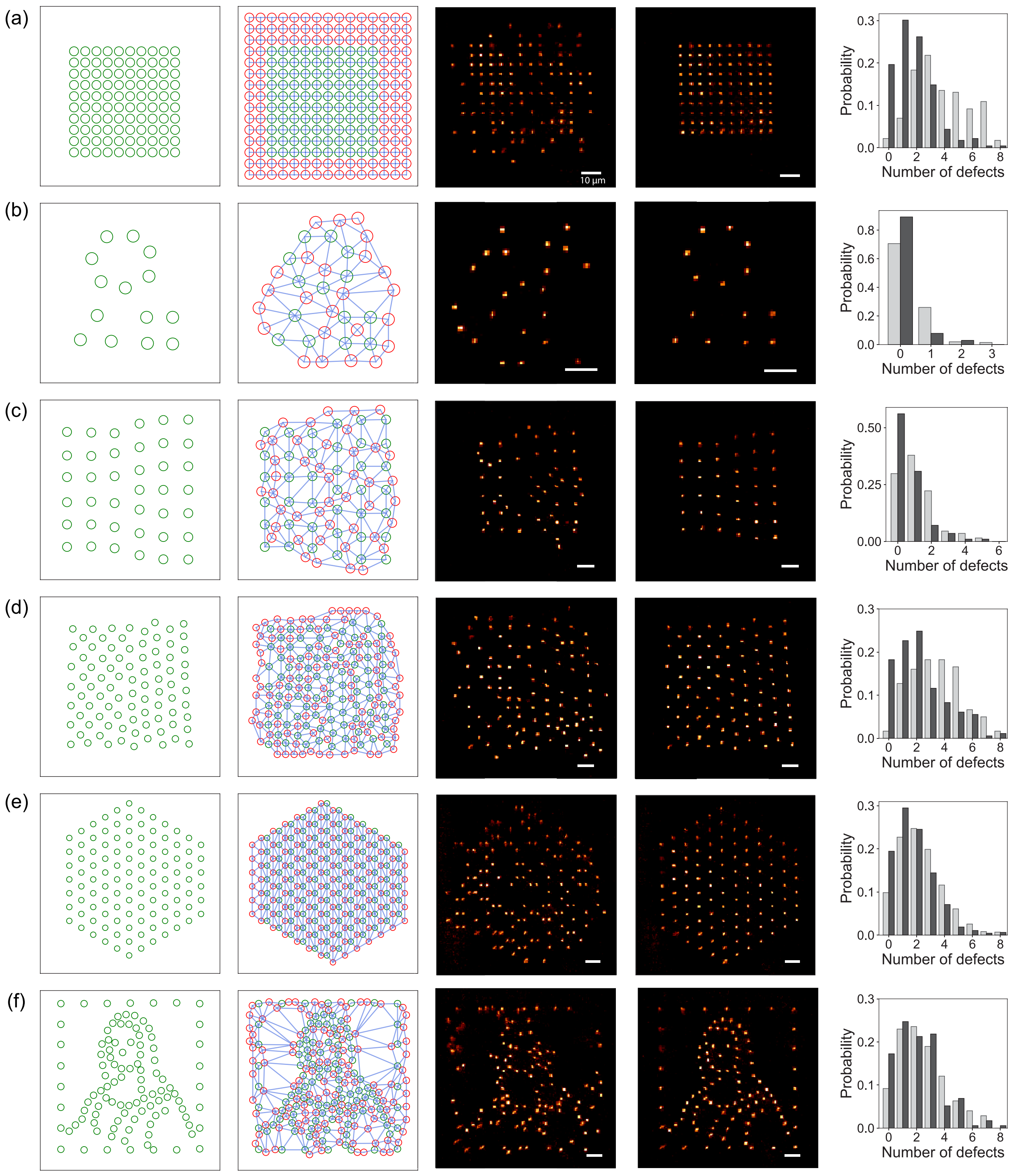

FIG. 10. Gallery of assembled arbitrary structures. Shown from left to right are the target structure, the structure with the generated reservoir traps (in red) and the allowed paths connecting traps, the fluorescence image of an initial random loading, the fluorescence image of the assembled structure, and the probability distribution of the number of defects after a rearrangement cycle (gray) and after two such cycles (dark gray). All white scale bars are 10 μm. (a) Compact square array ($N = 100$), (b) the arbitrary array used as an example in Sec. VI ($N = 14$), (c) an edge dislocation in a square lattice ($N = 39$), (d) a grain boundary between a square and a triangular lattice ($N = 91$), (e) a patch of a triangular lattice ($N = 108$), and (f) an atomic rendering of Mona Lisa ($N = 106$).

atoms, a high efficiency of a single rearrangement cycle is still essential as laser power is a limiting factor for scaling up the number of atoms. Figure 9(b) shows the probability distribution of the number of defects (missing atoms) after a single (left) or two (right) rearrangement cycles, showing the benefit of performing several cycles.

Examples of assembled structures of various types, with up to $N = 108$ atoms, can be seen in Fig. 10. The probability to have a given number of defects in the final array is shown in the histograms on the right, for a single rearrangement (gray) and for two cycles (dark gray). In the latter case, even for $N > 100$, defect-free arrays are obtained in about 20% of the shots. Using a trapping wavelength closer to resonance (820 nm) in order to generate more traps for a given laser power, we have been able to assemble arrays of up to 209 atoms without any given defects.

VIII. CONCLUSION

In this paper, we have shown how, without any change in the hardware used in [13], improved algorithms can significantly improve the capabilities of a moving-tweezer atom-by-atom assembler, both in terms of possible array geometries and in terms of achievable atom numbers due to the fact that fewer moves are required.

The algorithms demonstrated here can be used directly for the plane-by-plane assembly of three-dimensional structures [22]. Extending them to a full three-dimensional assembly with atoms being moved also longitudinally, along the lens optical axis, will require significant changes due to the fact that transverse moves (using an AOD) and longitudinal moves (done with an electrically tunable lens) do not obey the same constraints.

Another natural extension of this study, which we leave for future work, is to use multiple tweezers working in parallel, in the spirit of [14]. This approach should be particularly easy to adapt to the compression algorithm for assembling compact regular structures; then, assuming that the laser power for generating the multiple tweezers is not a limit, the assembly time could scale as $\sqrt{N}$, making it possible to assemble structures with several hundreds of atoms. Combined with other technical improvements, using, e.g., cryogenic environments to drastically extend the vacuum-limited lifetime, reaching a scale of 1000 atoms or more thus seems realistic in the relatively near future, which would open up a variety of exciting applications in quantum science and technology.

ACKNOWLEDGMENTS

We thank Loïc Henriet and Henrique Silvério for useful discussions and Gilles Kuhorn for contributions to testing some of the algorithms. This project received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 817482 (PASQuanS) and from the Région Île-de-France in the framework of DIM SIRTEQ (Project CARAQUES).

[1] A. Browaeys and T. Lahaye, Many-body physics with individually controlled Rydberg atoms, Nat. Phys. **16**, 132 (2020).

[2] H. Labuhn, D. Barredo, S. Ravets, S. de Léséleuc, T. Macrì, T. Lahaye, and A. Browaeys, Tunable two-dimensional arrays of single Rydberg atoms for realizing quantum Ising models, Nature (London) **534**, 667 (2016).

[3] H. Bernien, S. Schwartz, A. Keesling, H. Levine, A. Omran, H. Pichler, S. Choi, A. S. Zibrov, M. Endres, M. Greiner, V. Vuletić, and M. D. Lukin, Probing many-body dynamics on a 51-atom quantum simulator, Nature (London) **551**, 579 (2017).

[4] V. Lienhard, S. de Léséleuc, D. Barredo, T. Lahaye, A. Browaeys, M. Schuler, L.-P. Henry, and A. M. Läuchli, Observing the Space- and Time-Dependent Growth of Correlations in Dynamically Tuned Synthetic Ising Models with Antiferromagnetic Interactions, Phys. Rev. X **8**, 021070 (2018).

[5] H. Kim, Y. Park, K. Kim, H.-S. Sim, and J. Ahn, Detailed Balance of Thermalization Dynamics in Rydberg-Atom Quantum Simulators, Phys. Rev. Lett. **120**, 180502 (2018).

[6] A. Keesling, A. Omran, H. Levine, H. Bernien, H. Pichler, S. Choi, R. Samajdar, S. Schwartz, P. Silvi, S. Sachdev, P. Zoller, M. Endres, M. Greiner, V. Vuletić, and M. D. Lukin, Quantum Kibble-Zurek mechanism and critical dynamics on a programmable Rydberg simulator, Nature (London) **568**, 207 (2019).

[7] S. de Léséleuc, V. Lienhard, P. Scholl, D. Barredo, S. Weber, N. Lang, H. P. Büchler, T. Lahaye, and A. Browaeys, Observation of a symmetry-protected topological phase of interacting bosons with Rydberg atoms, Science **365**, 775 (2019).

[8] H. Levine, A. Keesling, A. Omran, H. Bernien, S. Schwartz, A. S. Zibrov, M. Endres, M. Greiner, V. Vuletić, and M. D. Lukin, High-Fidelity Control and Entanglement of Rydberg-Atom Qubits, Phys. Rev. Lett. **121**, 123603 (2018).

[9] H. Levine, A. Keesling, G. Semeghini, A. Omran, T. T. Wang, S. Ebadi, H. Bernien, M. Greiner, V. Vuletić, H. Pichler, and M. D. Lukin, Parallel Implementation of High-Fidelity Multiqubit Gates with Neutral Atoms, Phys. Rev. Lett. **123**, 170503 (2019).

[10] T. M. Graham, M. Kwon, B. Grinkemeyer, Z. Marra, X. Jiang, M. T. Lichtman, Y. Sun, M. Ebert, and M. Saffman, Rydberg-Mediated Entanglement in a Two-Dimensional Neutral Atom Qubit Array, Phys. Rev. Lett. **123**, 230501 (2019).

[11] I. S. Madjarov, J. P. Covey, A. L. Shaw, J. Choi, A. Kale, A. Cooper, H. Pichler, V. Schkolnik, J. R. Williams, and M. Endres, High-fidelity entanglement and detection of alkaline-earth Rydberg atoms, Nat. Phys. **16**, 857 (2020).

[12] N. Schlosser, G. Reymond, I. Protsenko, and P. Grangier, Sub-Poissonian loading of single atoms in a microscopic dipole trap, Nature (London) **411**, 1024 (2001).

[13] D. Barredo, S. de Léséleuc, V. Lienhard, T. Lahaye, and A. Browaeys, An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays, Science **354**, 1021 (2016).

[14] M. Endres, H. Bernien, A. Keesling, H. Levine, E. R. Anschuetz, A. Krajenbrink, C. Senko, V. Vuletic, M. Greiner, and M. D. Lukin, Atom-by-atom assembly of defect-free one-dimensional cold atom arrays, Science **354**, 1024 (2016).

[15] H. Kim, W. Lee, H.-g. Lee, H. Jo, Y. Song, and A. Jaewook, *In situ* single-atom array synthesis using dynamic holographic optical tweezers, Nat. Commun. **7**, 13317 (2016).

[16] M. O. Brown, T. Thiele, C. Kiehl, T.-W. Hsu, and C. A. Regal, Gray-Molasses Optical-Tweezer Loading: Controlling Collisions for Scaling Atom-Array Assembly, Phys. Rev. X **9**, 011057 (2019).

[17] W. Lee, H. Kim, and J. Ahn, Three-dimensional rearrangement of single atoms using actively controlled optical microtraps, Opt. Express **24**, 9816 (2016).

[18] D. Ohl de Mello, D. Schäffner, J. Werkmann, T. Preuschoff, L. Kohfahl, M. Schlosser, and G. Birkl, Defect-Free Assembly of 2D Clusters of more than 100 Single-Atom Quantum Systems, Phys. Rev. Lett. **122**, 203601 (2019).

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. (MIT Press, Cambridge, 2001).

[20] W. Lee, H. Kim, and J. Ahn, Defect-free atomic array formation using the Hungarian matching algorithm, Phys. Rev. A **95**, 053424 (2017).

[21] G. Călinescu, A. Dumitrescu, and J. Pach, LATIN 2006: Theoretical Informatics, in *Proceedings of the 7th Latin American Symposium, Valdivia, 2006*, edited by J. R. Correa, A. Hevia, and M. Kiwi, Lecture Notes in Computer Science Vol. 3887 (Springer, Berlin, 2006), pp. 262–273.

[22] D. Barredo, V. Lienhard, S. de Léséleuc, T. Lahaye, and A. Browaeys, Synthetic three-dimensional atomic structures assembled atom by atom, Nature (London) **561**, 79 (2018).

[23] D. F. Crouse, On implementing 2D rectangular assignment algorithms, IEEE Trans. Aerosp. Electron. Syst. **52**, 1679 (2016).

[24] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser *et al.*, SciPy 1.0: Fundamental algorithms for scientific computing in Python, Nat. Methods **17**, 261 (2020).

[25] H. Pichler, S.-T. Wang, L. Zhou, S. Choi, and M. D. Lukin, Quantum optimization for maximum independent set using Rydberg atom arrays, arXiv:1808.10816.

[26] L. Henriet, Robustness to spontaneous emission of a variational quantum algorithm, Phys. Rev. A **101**, 012335 (2020).

[27] F. P. Preparata and M. Shamos, *Computational Geometry: An Introduction* (Springer, New York, 1985).

[28] A. A. Hagberg, D. A. Schult, and P. J. Swart, in *Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, 2008*, edited by G. Varoquaux, T. Vaught, and J. Millman (unpublished), pp. 11–15.